matemática computacional I

Programar em Matlab

Universidade do Minho
Escola de Ciências
Departamento de Matemática
e Aplicações

maria irene falcão

2009/10

Prefácio

Estes textos têm como principal objectivo servir de apoio às aulas de Matemática Computacional I do 1º ano da Licenciatura em Matemática.

De entre os diversos "ambientes de computação" actualmente disponíveis, um dos mais frequentemente utilizados, para fins educacionais, é o MATLAB®. Este sistema apresenta características que justificam a sua ampla divulgação e utilização em diversos cursos de ciências e engenharia:

- é extremamente fácil de utilizar, sendo os dados introduzidos e manipulados de uma forma simples, especialmente no caso de vectores ou matrizes;
- inclui uma linguagem de programação de alto nível e bastante acessível, especialmente adequada a jovens que vão tomar um primeiro contacto com uma linguagem de programação;
- tem suporte em muitos sistemas computacionais diferentes, proporcionando, em grande medida, uma independência de plataforma. Programas escritos em MATLAB podem migrar para novas plataformas quando as necessidades do utilizador se alteram.
- tem um grande número de funções matemáticas pré-definidas que podem ser usados no âmbito de outras disciplinas da licenciatura em matemática, praticamente desde o início do curso;
- dispõe de boas capacidades gráficas, permitindo uma visualização, de forma simples, dos dados e resultados;
- é possível adquirir uma variedade de pacotes de programas (*toolboxes*) para fins mais específicos.

Em resumo, com o MATLAB os alunos podem realizar num ambiente agradável, tarefas de natureza diversa: desenvolvimento e análise de algoritmos, modelação, computação científica, visualização, etc.

Conteúdo

Matemática Computacional I

Iniciação ao Matlab

Conteúdo

1. Introdução

O sistema Matlab

O MATLAB é um sistema gráfico que integra a capacidade de realização de cálculos, programação e visualização gráfica num ambiente interactivo bastante agradável.

Este *software* é produzido pela companhia americana The Math Works, Inc. e o seu nome deriva do inglês "matrix laboratory".

O MATLAB trabalha com matrizes quadradas ou rectangulares, com elementos reais ou complexos, e derivou dos projectos LINPACK e EISPACK que são considerados como a origem de algum do melhor *software* numérico disponível para computação com matrizes. Para além da manipulação fácil de matrizes, a *package* permite o acesso a um número crescente de rotinas incorporadas, potencialidades gráficas a 2 e 3 dimensões e a possibilidade de configurar o *software* às necessidades de cada utilizador. O MATLAB possui ainda um conjunto de funções específicas - *toolboxes* - para uma dada aplicação e/ou tecnologia: Estatística e Análise de Dados, Optimização, Processamento de Sinal, Processamento de Imagem, etc.

O sistema MATLAB possui cinco partes principais:

1. As ferramentas para utilização e desenvolvimento;
2. A biblioteca de funções matemáticas;
3. A linguagem de programação (de alto nível);
4. As funções gráficas;
5. As bibliotecas de interfaces.

As versões actuais do MATLAB permitem realizar muitas tarefas recorrendo às suas *interfaces* gráficas - menus, botões, etc. Na maior parte dos casos o recurso a estas potencialidades pode ser feito de uma forma natural e intuitiva, pelo que os presentes textos darão mais ênfase aos comandos *escritos*.

Quando se invoca o MATLAB[1], é criada uma janela com várias "subjanelas" e menus. Este ambiente de trabalho pode ser personalizado, de acordo com as preferências do utilizador.

Do lado direito do ecrã aparece, por defeito, a janela de comandos - *Command Window*, através da qual é possível comunicar com o interpretador do MATLAB. Quando aparece o símbolo >>, o MATLAB está pronto a receber instruções. A janela de comandos pode ser apagada através do comando **clc** ou usando a opção **Clear Command Window** do menu **Edit**.

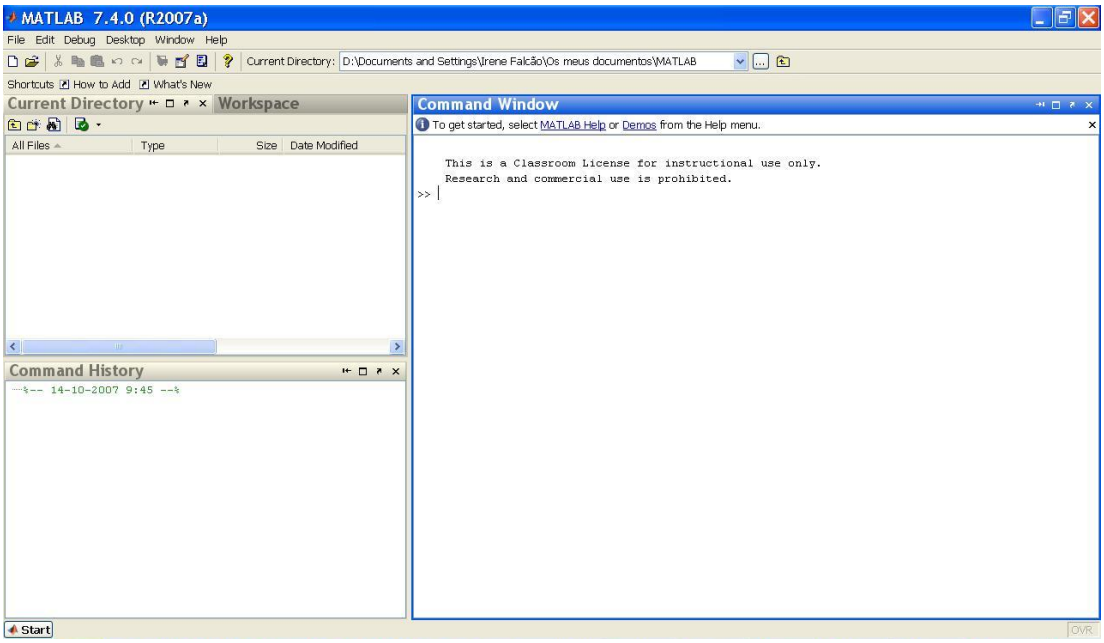


Figura 1.1: O ambiente de trabalho do MATLAB

No lado superior esquerdo pode ver-se a janela *Current Directory* onde pode ser obtida informação relativa aos ficheiros da pasta de trabalho do Matlab. Em geral, o MATLAB só reconhece ficheiros que estejam nessa pasta ou no caminho de pesquisa do sistema -*path*. Os menus e botões desta janela podem ser usados para examinar os ficheiros, ou alternativamente, podem ser usados comandos equivalentes na janela de comandos: **pwd** - retorna o nome da pasta corrente; **cd** - altera a pasta de trabalho; **dir** - lista todos os ficheiros da pasta corrente; **what** - lista apenas os ficheiros MATLAB. Os comandos **delete** e **type** permitem apagar um ficheiro e mostrar na janela de comandos um ficheiro, respectivamente.

```
>> pwd
ans = D:\Irene\0-Aulas\MComp\MATLAB
>> dir
.                AritmeticaC.pdf  teste.m          variaveis.mat ..
folha1.pdf       teste2.m

>> what
```

[1] A versão utilizada nestes textos é a versão 7.4.0(R2007a)

```
M-files in the current directory D:\Irene\0-Aulas\MComp\MATLAB
teste teste2

MAT-files in the current directory D:\Irene\0-Aulas\MComp\MATLAB
variaveis
```

Todas as variáveis usadas na sessão de trabalho são gravadas no espaço de trabalho - *MATLAB Workspace*.

O conteúdo de *Workspace* pode também ser visto através do comando **whos** ou **who**.

```
>> whos
  Name        Size              Bytes  Class     Attributes
  a           1x1                  16  double    complex
  b           3x2                  48  double
  c           1x1                   8  double

>> who
Your variables are: a  b  c
```

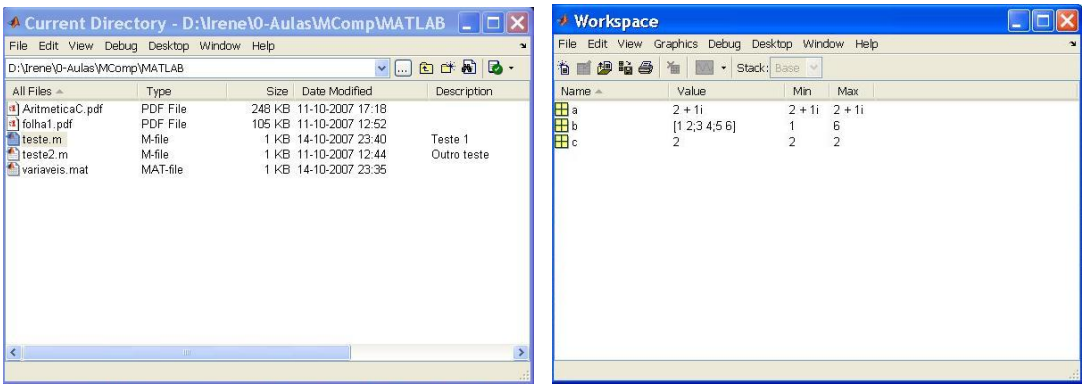


Figura 1.2: As janelas *Current Directory* e *Workspace*

O texto de cada sessão de trabalho pode ser guardado através do comando **diary**. Por exemplo, todo o texto relativo à sessão que se inicia imediatamente a seguir à instrução

```
>> diary sessao
```

e termina com o comando

```
>> diary off
```

é guardado num ficheiro de texto chamado *sessao*.

As variáveis definidas ou obtidas numa sessão de trabalho podem também ser guardadas num ficheiro para serem posteriormente usadas, recorrendo aos comandos **save** e **load**. Assim, o comando

```
>> save variaveis
```

guarda todas as variáveis do espaço de trabalho num ficheiro chamado *variaveis.mat*. Para recuperar estes valores basta fazer

```
>> load variaveis
```

É ainda possível guardar apenas algumas variáveis. Por exemplo, o comando

```
>> save apenas X Y
```

guarda as variáveis X e Y num ficheiro chamado *apenas.mat*.

Para apagar a variável X, pode usar-se o comando **clear** X. Todas as variáveis do espaço de trabalho serão apagadas, se for usado apenas **clear**.

Alternativamente, podem ser usados os botões da janela *Workspace* para realizar estas tarefas.

Finalmente, na janela *Command History*, no lado inferior esquerdo do ambiente de trabalho do MATLAB, pode ver-se um historial de todas as sessões de trabalho realizadas, desde que se apagou pela última vez esta informação (opção **Clear Command History** do menu **Edit**). Verifique o que acontece se seleccionar uma das linhas do *Command History*!

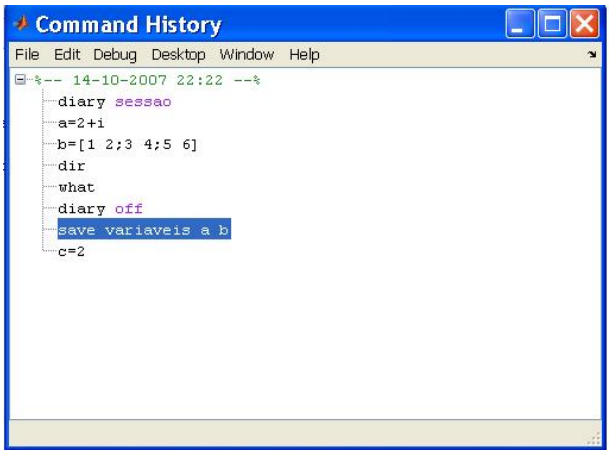


Figura 1.3. Command History do MATLAB

As declarações no MATLAB são frequentemente da forma

variavel=*expressão*

ou simplesmente

expressão.

No primeiro caso, o valor de *expressão* é atribuído à variável *variavel* para uso futuro. Quando o nome da variável é omitido (assim como o sinal =), o MATLAB cria automaticamente uma variável com o nome *ans* (de *answer*). Por exemplo, as declarações

```
>> a=2*3
```

e

```
>> 2*4
```

originam, respectivamente

```
a =
     6

ans =
     8
```

O nome de uma variável deve sempre começar por uma letra, seguido de um conjunto de letras ou números (ou ainda o sinal _), até ao máximo de 31 caracteres. O MATLAB **distingue as letras minúsculas das maiúsculas**. Assim, a e A não representam a mesma variável. **Todas as funções do MATLAB têm nomes escritos em minúsculas** (ver Secção 1.2).

Sempre que se pretenda que o resultado de uma operação ou atribuição não apareça no ecrã (evitando assim o aparecimento de uma quantidade de informação pouco útil), deve usar-se o símbolo ;. A declaração

```
>> a=2*3;
```

atribui o valor 6 à variável a, mas não mostra no ecrã o resultado dessa atribuição.

Quando a expressão é muito complicada, necessitando de mais de uma linha para ser definida, deve usar-se o símbolo ... antes de mudar de linha, para indicar que a expressão continua.

```
>> 2*cos(3)+ 3*cos(2)-5*cos(1)+ 8*cos(5)-0.5*i*cos(4)+...
2*sin(3)+ 3*i*sin(2)-5*sin(1);
```

Em contrapartida, se as expressões forem muito simples, podem ser definidas simultaneamente na mesma linha de comandos, usando o símbolo ,.

```
>> a=2*3,b=2*4
a =
     6

b =
     8
```

Definição e manipulação de matrizes

O MATLAB trabalha essencialmente com um tipo de objecto, uma matriz numérica rectangular com elementos eventualmente complexos. Para o MATLAB, uma matriz é um *array*[2] bidimensional; escalares são entendidos como matrizes 1×1; vectores são matrizes $1 \times n$ ou $n \times 1$. Assim, as operações e comandos em MATLAB devem ser entendidos duma forma natural, enquanto operações entre matrizes.

A maneira mais simples de definir uma matriz pequena é introduzir explicitamente os seus elementos da seguinte forma: cada elemento da matriz é separado pelo símbolo , (ou espaço) e cada

[2] um *array* é um conjunto de dados que podem ser acedidos por indexação. No MATLAB, o conjunto de índices é sempre uma sequência de inteiros que começa em 1

linha da matriz é separada por ; (ou mudança de linha). Os elementos da matriz devem estar rodeados pelos símbolos [e] .

Por exemplo as atribuições

```
>> A=[1,1,1;2,2,2;3,3,3];
>> B=[ 1 1 1
       2 2 2
       3 3 3];
>> C=[1 1 1;2 2 2;3 3 3];
```

produzem exactamente a mesma matriz:

```
     1     1     1
     2     2     2
     3     3     3
```

Para obter um determinado elemento da matriz A, basta indicar a sua linha e coluna. Por exemplo,

```
>> A(2,3)
ans =
     2
```

Uma das vantagens do MATLAB é a de não ser necessário dimensionar as matrizes *a priori*. A cada nova instrução é feito o redimensionamente da matriz. Assim, as declarações

```
>> A(5,5)=1,B(5,1)=1
```

produzem as novas matrizes

```
A =
     1     1     1     0     0
     2     2     2     0     0
     3     3     3     0     0
     0     0     0     0     0
     0     0     0     0     1

B =
     1     1     1
     2     2     2
     3     3     3
     0     0     0
     1     0     0
```

Outra forma simples de obter matrizes maiores à custa de matrizes menores já definidas é a seguinte: se pretendermos acrescentar à matriz A inicial a linha 4 4 4, basta fazer

```
>> l=[4 4 4];A=[A;l]
```

ou apenas

```
>> A=[A;4 4 4]
```

para se obter a nova matriz

```
A =
     1     1     1
     2     2     2
     3     3     3
     4     4     4
```

Para obter submatrizes de uma dada matriz deve usar-se o símbolo : para indicar quais as linhas e colunas da matriz inicial a considerar. Por exemplo, considerando a matriz

```
>> A=[1  2  3
      4  5  6
      7  8  9
     -1 -2 -3]
```

a declaração

```
>> B=A(2:3,1:3)
```

resultará numa matriz que contém as linhas 2 e 3 e as colunas 1 a 3 da matriz inicial,

```
B =
     4     5     6
     7     8     9
```

enquanto a atribuição

```
>> C=A(:,1:2)
```

resultará numa matriz que tem as mesmas linhas e as 2 primeiras colunas de A.

```
C =
     1     2
     4     5
     7     8
    -1    -2
```

Pode ainda usar-se **end** para referir o índice mais elevado de uma dimensão. Por exemplo,

```
>> A(end-1,end)
```

resultará no elemento da matriz A que está na penúltima linha e última coluna, isto é,

```
ans =
     9
```

Também é possível apagar linhas e colunas de uma matriz, usando os símbolos [ ]. Para retirar a segunda linha da matriz A definida anteriormente basta fazer

```
>> A(2,:)=[]

A =
     1     2     3
     7     8     9
    -1    -2    -3
```

Vectores com componentes inteiras podem também ser usados como índices. Por exemplo,

```
>> D=A([1 3],:)
```

produzirá uma matriz D cujas linhas são as linhas 1 e 3 da matriz A.

```
 D =
     1     2     3
    -1    -2    -3
```

Além disso, vectores como índices podem aparecer em ambos os lados de uma atribuição. Por exemplo, sendo E a matriz

```
>> E=[1 1 1
      2 2 2
      3 3 3];
```

a atribuição

```
>> A([1 3],:)=E([2 3],:)
```

resultará na matriz

```
 A =
     2     2     2
     7     8     9
     3     3     3
```

O símbolo : permite ainda definir vectores e/ou matrizes de modo muito simples. Por exemplo,

```
>> x=-2:1:2

x =
    -2    -1     0     1     2

>> y=[0:3:9;2:2:8;5:5:20]

y =
     0     3     6     9
     2     4     6     8
     5    10    15    20
```

O MATLAB usa a notação decimal convencional para representar números, sendo também possível incluir-se, na representação de um número, potências de base 10. Assim, as expressões 0.001 e 1E-3 representam o mesmo número. A notação usada para a unidade imaginária é o i ou j. As expressões 1+2i e 1+2j representam o número complexo cuja parte real é 1 e a parte imaginária é 2.

As expressões podem ser construídas usando os operadores aritméticos usuais e correspondentes precedências. Assim, tem-se

- `+` adição
- `-` subtracção
- `*` multiplicação
- `/` divisão à direita [3]
- `\` divisão à esquerda
- `^` potenciação

O MATLAB tem incorporadas funções matemáticas elementares tais como **abs**, **sqrt**, **log**, etc. Para uma lista completa destas e outras funções, veja a secção seguinte

Por defeito, o MATLAB trabalha no sistema de numeração de norma IEEE em formato duplo, isto é, no sistema $F(2, 53, -1021, 1024)$[4]. Para controlar o formato de saída dos resultados pode usar-se o comando **format**. Por defeito, o MATLAB apresenta os resultados no **format short** que corresponde ao uso de 5 dígitos. É possível alterar este formato, usando formatos que permitem mais dígitos ou usam a notação científica, como se exemplifica na tabela abaixo. Para outras opções, use o *help* do MATLAB (**help format**).

Formato	Descrição
format	o mesmo que short
format short	notação de vírgula fixa, com 5 dígitos
format long	notação de vírgula fixa, com 15 dígitos
format short e	notação de vírgula flutuante, com 5 dígitos
format long e	notação de vírgula flutuante, com 15 dígitos
format short g	o melhor dos formatos de vírgula fixa ou flutuante, com 5 dígitos
format long g	o melhor dos formatos de vírgula fixa ou flutuante, com 15 dígitos

Por exemplo:

```
>> X=[4/3 sqrt(2)/2.5E4]
X =
    1.3333    0.0001

>> format short;X
X =
    1.3333    0.0001
```

[3] As operações com matrizes tornam conveniente o uso de dois símbolos para a divisão; ver Secção 1.2.

[4] Relembre a matéria já leccionada sobre este tema

```
>> format short e;X
X =
  1.3333e+000  5.6569e-005

>> format short g;X
X =
       1.3333  5.6569e-005

>> format long;X
X =
   1.33333333333333   0.00005656854249
```

2. Operações com matrizes

Transposta de uma matriz

O símbolo ’ denota a transposta de uma matriz, no caso em que esta é real, ou a transconjugada de uma matriz, se esta for complexa.[5] Assim, as declarações

```
>> A=[1 2 3;4 5 6;7 8 9]
>> B=A'
>> X=[-1+i 2-i 3]'
```

resultam nas matrizes

```
A =

     1     2     3
     4     5     6
     7     8     9
B =

     1     4     7
     2     5     8
     3     6     9
X =

  -1.0000 - 1.0000i
   2.0000 + 1.0000i
   3.0000
```

[5] A transposta de uma matriz complexa A pode obter-se fazendo `A.'` ou `conj(A')`.

Soma de matrizes

Os símbolos $+$ e $-$ designam soma e subtracção de matrizes e estas operações estão definidas de forma usual. Assim, a atribuição

```
>> C=A+B
```

resulta na matriz

```
C =
     2     6    10
     6    10    14
    10    14    18
```

No MATLAB é ainda possível definir a operação $A+B$ ou $A-B$ sempre que uma das matrizes tem ordem 1, i.e. é um escalar. Neste caso, a matriz resultante é a que se obtém somando ou subtraindo o escalar a todos os elementos da outra matriz. As declarações

```
>> D=A-1;
>> D=-1+A
```

originam exactamente a mesma matriz

```
D =
     0     1     2
     3     4     5
     6     7     8
```

Produto de matrizes

O símbolo * denota multiplicação de matrizes e esta operação está definida da forma usual. Por exemplo, se

$$A=\begin{pmatrix}1&2&3\\4&5&6\\7&8&9\end{pmatrix},\qquad B=\begin{pmatrix}1&2\\3&4\\5&6\end{pmatrix},\qquad x=\begin{pmatrix}1&2&3\end{pmatrix},\qquad y=\begin{pmatrix}4&5&6\end{pmatrix},$$

então as atribuições

```
>> M1=A*B
>> M2=4*A
>> M3=x'*y
```

originam as matrizes

```
M1 =
    22    28
    49    64
    76   100
M2 =
     4     8    12
    16    20    24
    28    32    36
M3 =
     4     5     6
     8    10    12
    12    15    18
```

e os comandos

```
>> M4=B*A;
>> M5=x*y
>> M6=A*x
```

produzem

```
??? Error using ==> mtimes
Inner matrix dimensions must agree.
```

Quociente de matrizes

No MATLAB existem dois símbolos para representar a "divisão" de duas matrizes: $\backslash$ e $/$. Se A é uma matriz quadrada invertível, então $A\backslash B$ e B/A correspondem formalmente ao produto à esquerda e à direita, de B pela inversa de A, i.e.

$$A\backslash B = A^{-1} * B$$

$$B/A = B * A^{-1}$$

Como seria de esperar, o resultado destas operações não passa pelo cálculo explícito da inversa de A, mas sim pela resolução de sistemas[6]. Em geral

$$X = A\backslash B \text{ é solução da equação } A * X = B$$

$$Y = B/A \text{ é solução da equação } Y * A = B$$

Sejam

$$A = \begin{pmatrix} 1 & 2 & 0 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix} \quad \text{e} \quad B = \begin{pmatrix} 3 & -3 & 1 \\ 15 & 12 & 13 \\ 24 & 18 & 19 \end{pmatrix}$$

Então, se

[6] Este problema será estudado com detalhe na disciplina de Álgebra Linear e Geometria Analítica I

```
>> X=A\B
```

obtém-se

```
 X =
    1.0000    1.0000   -1.0000
    1.0000   -2.0000    1.0000
    1.0000    3.0000    2.0000
```

Naturalmente que se as matrizes forem de ordem 1, i.e. escalares, as duas divisões correspondem à divisão usual. Assim, $2\backslash 8 = 8/2 = 4$ e $8\backslash 2 = 2/8 = 0.25$.

Operações elemento a elemento

As operações elemento a elemento diferem das operações usuais com matrizes, mas podem ter grande aplicação prática. Para indicar que uma dada operação é para ser feita elemento a elemento deve usar-se o ponto (.) imediatamente antes do operador.

Por exemplo, se

$$A = X.\hat{}Y,\ B = X.*Y \ \text{ e } \ C = X./Y,$$

então

$$a_{ij} = (x_{ij})^{y_{ij}},\ b_{ij} = x_{ij} * y_{ij} \ \text{ e } c_{ij} = x_{ij}/y_{ij}.$$

Para que estas operações possam ser executadas, as matrizes (ou vectores) X e Y têm que ter a mesma ordem. Note-se que $X. + Y$ e $X. - Y$ não estão definidas (Porquê?).

Consideremos as matrizes

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \qquad \text{e} \qquad Y = \begin{pmatrix} 2 & 3 \\ 3 & 0 \end{pmatrix}$$

As seguintes operações elemento a elemento

```
>> E1=X.*Y,E2=Y./X,E3=X.^Y
```

produzem as matrizes

```
E1 =
     2     6
     9     0
E2 =
    2.0000    1.5000
    1.0000         0
E3 =
     1     8
    27     1
```

O MATLAB tem incorporadas algumas funções muito úteis que permitem definir matrizes muito usadas. Para definir, por exemplo a matriz identidade, a matriz nula ou a matriz "constante" podem usar-se as funções **eye**, **zeros** ou **ones**[7], cujo modo de utilização pode ser obtido invocando o **help** do MATLAB.

```
>> help eye
 EYE Identity matrix.
    EYE(N) is the N-by-N identity matrix.
    EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1's on
    the diagonal and zeros elsewhere.
    EYE(SIZE(A)) is the same size as A.
                ...
>> help zeros
 ZEROS  Zeros array.
    ZEROS(N) is an N-by-N matrix of zeros.
    ZEROS(M,N) or ZEROS([M,N]) is an M-by-N matrix of zeros.
    ZEROS(M,N,P,...) or ZEROS([M N P ...]) is an M-by-N-by-P-by-...
    array of zeros.
    ZEROS(SIZE(A)) is the same size as A and all zeros.
                ...
>> help ones
 ONES   Ones array.
    ONES(N) is an N-by-N matrix of ones.
    ONES(M,N) or ONES([M,N]) is an M-by-N matrix of ones.
    ONES(M,N,P,...) or ONES([M N P ...]) is an M-by-N-by-P-by-...
    array of ones.
    ONES(SIZE(A)) is the same size as A and all ones.
                 ...
```

As declarações

```
>> A1=eye(3)
>> A2=zeros(2,3)
>> A3=2*ones(size(A2))
```

produzem as matrizes

```
A1 =
     1     0     0
     0     1     0
     0     0     1
```

[7] O nome das funções, tal como já foi referido, deve ser escrito em minúsculas.

```
A2 =
     0     0     0
     0     0     0

A3 =
     2     2     2
     2     2     2
```

Operadores relacionais

No MATLAB, os operadores relacionais são os seguintes:

Operadores relacionais	Descrição
$==$	igual
$<=$	menor ou igual
$>=$	maior ou igual
$\sim=$	diferente
$<$	menor
$>$	maior

O resultado de um operador relacional é do tipo lógico - **logical** - e pode ser **1** (verdadeiro) ou **0** (falso). Assim, $2 > 3$ é 0 e $2 == 4 - 2$ é 1.

Adicionalmente podem ainda formar-se expressões lógicas mais complicadas combinando elementos lógicos e expressões de relação, com os seguintes operadores lógicos.

Operadores lógicos	Descrição
&	e
|	ou
$\sim$	negação
xor	ou exclusivo

Por exemplo, ~ 0 é 1, ~ 1 é 0, ~ 5 é 0, 0|4 é 1, 0&4 é 0 e 2&3 é 1. O resultado da aplicação de um operador relacional a matrizes da mesma dimensão é uma matriz com 0's e 1's dependendo do valor das expressões nas posições correspondentes. Por exemplo,

```
>> A=[1 2;3 4]; B=[2 2;1 5];
>> A>=2

ans =

     0     1
     1     1
```

```
>> A>B
ans =
     0     0
     1     0
```

Arrays lógicos podem também ser usados como índices. Obtém-se neste caso um array com os valores para os quais o índice é 1. Por exemplo,

```
>> a=[1 2 3 4];
>> a>2
ans =
     0     0     1     1
>> a(ans)
ans =
     3     4
```

Note-se que as instruções

```
>> b=[0 0 1 1];
>> a(b)
```

resultam em

```
 ??? Subscript indices must either be real positive
integers or logicals.
```

porque b é um *array* de reais. Teria que ser convertido num *array* lógico para que a operação fosse possível.

```
>> a(logical(b))
ans =
     3     4
```

Outros operadores lógicos, como && e || e funções, como por exemplo, **any**, **all** e **find** serão abordados posteriormente.

Funções

O MATLAB contém um conjunto grande de funções já definidas. Algumas dessas funções são funções matriciais, como por exemplo $\mathbf{inv}(A)$, que calcula a inversa de uma matriz quadrada A (invertível), outras estão definidas elemento a elemento. Por exemplo, $C = \cos(A)$ é uma matriz cujos elementos c_{ij} são os valores do cosseno dos elementos de $A = (a_{ij})$, i.e. $c_{ij} = \cos(a_{ij})$.

Segue-se, a título de exemplo, uma lista das funções mais usadas, dividida em 5 categorias: funções matemáticas elementares, matrizes elementares e manipulação de matrizes, funções matriciais, polinómios e análise de dados. Algumas destas funções são ainda *estranhas* aos alunos

do 1º ano. Ao longo do ano, elas poderão vir a ser especialmente úteis e apreciadas nas várias disciplinas do curso. Uma lista completa de todas as funções existentes em cada uma destas categorias pode ser obtida invocando o **help**: `help elfun`, `help elmat`, `help matfun`, `help polyfun`, `help datafun`.[8]

➠ Funções Matemáticas Elementares

➠ Funções Trigonométricas

sin - *Seno*.
sinh - *Seno hiperbólico*.
asin - *Arco-seno*.
asinh - *Arco-seno hiperbólico*.
cos - *Cosseno*.
cosh - *Cosseno hiperbólico*.
acos - *Arco-cosseno*.
acosh - *Arco-cosseno hiperbólico.*
tan - *Tangente*.
tanh - *Tangente hiperbólica*.
atan - *Arco-tangente*.
atanh - *Arco-tangente hiperbólica*.
sec - *Secante*.
sech - *Secante hiperbólica*.
asec - *Arco-secante*.
asech - *Arco-secante hiperbólica*.
csc - *Cossecante*.
csch - *Cossecante hiperbólica*.
acsc - *Arco-cossecante*.
acsch - *Arco-cossecante hiperbólica*.
cot - *Cotangente*.
coth - *Cotangente hiperbólica*.
acot - *Arco-cotangente*.
acoth - *Arco-cotangente hiperbólica*.

➠ Função Exponencial

exp - *Exponencial*.
log - *Logaritmo neperiano*.
log10 - *Logaritmo na base 10*.
sqrt - *Raiz quadrada*.

➠ Funções Complexas

abs - *Módulo*.
conj - *Conjugado*.
imag - *Parte imaginária*.
real - *Parte real*.

[8] As diversas categorias de funções encontram-se em MATLAB7\TOOLBOX\MATLAB.

➠ Matrizes Elementares e Manipulação de Matrizes

➟ Matrizes Elementares

zeros - *Matriz nula.*
ones - *Matriz com todos os elementos 1.*
eye - *Matriz identidade.*

➟ Variáveis especiais e constantes

ans - *Resposta mais recente.*
eps - *epsilon da máquina.*
realmax - *Maior número em vírgula flutuante.*
realmin - *Menor positivo em vírgula flutuante.*
pi - *3.1415926535897....*
i, j - *Unidade imaginária.*
inf - *Infinito.*
NaN - *Not-a-Number.*

➟ Manipulação de Matrizes

diag - *Criar ou extrair diagonais.*
reshape - *Redimensionar uma matriz.*
tril - *Extrair parte triangular inferior.*
triu - *Extrair parte triangular superior.*

➠ Funções Matriciais

➟ Análise de Matrizes

cond - *Número de condição de uma matriz.*
norm - *Norma matricial ou vectorial.*
rank - *Característica de uma matriz.*
det - *Determinante de uma matriz.*
trace - *Traço de uma matriz.*
null - *Núcleo de uma matriz.*

➟ Equações Lineares

\ e / - *Solução de um sistema linear; use "help slash".*
chol - *Decomposição de Cholesky.*
linsolve - *Solução de um sistema linear.*
lu - *Decomposição LU.*
inv - *Inversa de uma matriz.*

➟ Valores Próprios

eig - *Valores e vectores próprios.*
poly - *Polinómio característico.*
hess - *Forma de Hessenberg.*

➟ Funções

expm - *Exponencial de uma matriz.*
logm - *Logaritmo de uma matriz.*
sqrtm - *Raiz quadrada de uma matriz.*

➠ **Polinómios**

roots	- *Raízes de um polinómio.*
poly	- *Construção de um polinómio, dadas as suas raízes.*
polyval	- *Avaliar um polinómio.*
polyvalm	- *Avaliar um polinómio, cujo argumento é uma matriz.*
polyder	- *Derivada de um polinómio.*
conv	- *Produto de polinómios.*
deconv	- *Divisão de polinómios.*

➠ **Análise de Dados**

max	- *Máximo.*
min	- *Mínimo.*
mean	- *Média.*
median	- *Mediana.*
std	- *Desvio padrão.*
sort	- *Ordenação (ascendente).*
sum	- *Soma dos elementos.*
prod	- *Produto dos elementos.*
cross	- *Produto vectorial.*
dot	- *Produto escalar.*

Para conhecer o modo de utilização de cada função, basta fazer **help** seguido do nome da função pretendida. Por exemplo,

```
>> help triu
```

indica

```
 TRIU Extract upper triangular part.
    TRIU(X) is the upper triangular part of X.
    TRIU(X,K) is the elements on and above the K-th diagonal of
    X.  K = 0 is the main diagonal, K > 0 is above the main
    diagonal and K < 0 is below the main diagonal.

    See also TRIL, DIAG.
```

Outra facilidade do MATLAB pode ser obtida recorrendo ao uso de **lookfor** *palavra* que permite encontrar todas as funções que contêm a palavra *palavra* na primeira linha de comentário. Por exemplo,

```
>> lookfor upper
```

tem como resultado

```
TRIU Extract upper triangular part.

UPPER  Convert string to uppercase.
```

3. *Notebook*

O *Notebook* do MATLAB permite aceder às potencialidades numéricas e gráficas do MATLAB a partir de um processador de texto - Microsoft Word. [9]

Usando o *Notebook* , é possível criar um documento contendo texto, comandos MATLAB e os resultados da execução desses comandos.

Nesta secção, iremos apenas referir as instruções básicas para começar a usar o *Notebook* . O guia do utilizador *MATLAB Notebook User's Guide* contém uma descrição detalhada das capacidades do *Notebook* .

Criar e editar um *M-book*

Um documento *Notebook* é chamado um *M-book* . Para criar um *M-book* , basta executar o Word, escolher **New** no menu **File** do Word e seleccionar o modelo *M-book* .

O Word cria então um novo documento usando o modelo *M-book* e abre o MATLAB (se este não se encontrar activado). Na barra do menu do Word aparece um novo menu chamado **Notebook** que contém comandos para o *Notebook* .

Para editar um *M-book* já existente, basta escolher o comando **Open** no menu **File** do Word e seleccionar o documento *M-book* desejado.

Conteúdo de um *M-book*

O texto e os comandos MATLAB que constituem um *M-book* podem ser escritos da mesma forma que qualquer texto de outro documento Word. O estilo do documento pode, por isso, ser alterado de acordo com o gosto pessoal do utilizador.

O *Notebook* usa células de entrada (*input cells*) para definir comandos MATLAB . Estas células de entrada podem ser constituídas por um comando de uma ou mais linhas e por texto.

Definir e Executar Comandos MATLAB

1. Escreva o comando como texto, deixando o cursor no final da linha de texto. (Não use a tecla **Enter** ou **Return**.)
2. Seleccione o comando **Evaluate Cell** do menu do **Notebook** ou pressione **Ctrl**-**Enter**.

O *Notebook* define, então, o comando MATLAB introduzido como uma célula de entrada e cria uma célula de saída (*output cell*) com o resultado da execução do comando. Os caracteres de uma célula de entrada aparecem com a cor verde e os de uma célula de saída, com a cor azul. O início e fim de células de entrada/saída estão assinalados com os marcadores [e], respectivamente.

O *Notebook* permite ainda definir uma sequência de comandos MATLAB e avaliá-la como um grupo (*cells group*).

[9] O *Notebook* do MATLAB suporta as versões Word 2000, e 2002 e 2003 (ambas para XP).

Definir e Executar uma Sequência de Comandos MATLAB

1. Escreva a sequência de comandos a executar.
2. Seleccione com o rato todo o conjunto de comandos.
3. Pressione **Ctrl-Enter**.

Exemplo 1

1. Escreva o texto

```
a=[1 2 3;4 5 6;7 8 9]
```

2. Pressione **Ctrl-Enter** para obter

```
[ a=[1 2 3;4 5 6;7 8 9] ]
[a =
     1     2     3
     4     5     6
     7     8     9]
```

Exemplo 2

1. Escreva o texto

```
a=[1 2 3;4 5 6;7 8 9]
b=2*a
b(3,3)=0
```

2. Seleccione todo o texto com o rato.

3. Pressione **Ctrl-Enter** para obter

```
[ a=[1 2 3;4 5 6;7 8 9]
 b=2*a
 b(3,3)=0 ]
[b =
     2     4     6
     8    10    12
    14    16    18
 b =
     2     4     6
     8    10    12
    14    16     0 ]
```

4. Exercícios

Exercício 1. Defina, no MATLAB, a matriz

$$A = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & -1 \\ 1 & -5 & 4 \end{pmatrix}$$

e obtenha, a partir de A, as seguintes matrizes.

$$A_1 = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & 1 \\ 1 & 5 & 4 \end{pmatrix} \quad A_2 = \begin{pmatrix} 1 & -1 & 2 \\ 0 & 2 & 1 \\ 0 & 0 & 4 \end{pmatrix} \quad A_3 = \begin{pmatrix} 1 & -1 & 2 & 1 \\ 3 & 2 & -1 & 2 \\ 1 & -5 & 4 & 3 \end{pmatrix}$$

$$A_4 = \begin{pmatrix} 1 & -1 & 2 \\ 3 & 2 & -1 \\ 1 & -5 & 4 \\ 1 & 2 & 3 \end{pmatrix} \quad A_5 = \begin{pmatrix} 2 & 1 & -1 & 2 \\ 0 & 1 & -1 & 2 \\ 0 & 3 & 2 & -1 \\ 0 & 1 & -5 & 4 \end{pmatrix} \quad A_6 = \begin{pmatrix} 1 & -1 & 2 & 0 & 0 \\ 3 & 2 & -1 & 0 & 0 \\ 1 & -5 & 4 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \end{pmatrix}$$

$$A_7 = \begin{pmatrix} 1 & -1 & 2 & 1 & -1 & 2 \\ 3 & 2 & 1 & 3 & 2 & 1 \\ 1 & 5 & 4 & 1 & 5 & 4 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \quad A_8 = \begin{pmatrix} 1 & -1 \\ 3 & 2 \end{pmatrix}.$$

Exercício 2. Construa o vector `z=[10,20,30,40,50,60,70,80]`. Indique que vectores se obtêm se efectuarmos cada um dos seguintes comandos:

```
a. >> u=z(1:2:7)              b. >> v=z(7:-2:1)
c. >> w=z([3 4 8 1])          d. >> z(1:2:7)=zeros(1,4)
e. >> z(7:-2:1)=ones(1,4)     f. >> z(1:3)=[ ]
```

Exercício 3. Dada a matriz `A = [2 7 9 7;3 1 5 6;8 1 2 5]`, explique o resultado dos seguintes comandos:

```
a. >> A(1,[2 3])              b. >> A(:,[1 4])
c. >> A([2 3],[3 1])          d. >> reshape(A,2,6)
e. >> A(:)                    f. >> A(end,:)
g. >> A(1:3,:)                h. >> [A ;A(1:2,:)]
i. >> sum(A)                  j. >> sum(A')
k. >> sum(A,2)                l. >> [[A;sum(A)] [sum(A,2);sum(A(:))]]
```

Exercício 4. Defina (de uma forma simples) uma matriz A:

a) de ordem 5 com todos os elementos iguais a 3;

b) diagonal, de ordem 5, com todos os elementos da diagonal iguais a 8;

c) de ordem 4 em que cada coluna seja o vector $v = (1, 2, 3, 4)^{\mathsf{T}}$;

d) com a seguinte estrutura

$$\begin{pmatrix} 1 & 1 & 1 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 2 & 0 \\ 1 & 1 & 1 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 \end{pmatrix}.$$

Exercício 5. Defina as matrizes

```
>> x=[1 4 2]
>> y=[-1+i -i 1+i]
>> A=[1 2 3;-1 3 -2]
>> B=[1 -1 1;0 1 2; 1 1 1]
```

Verifique quais das seguintes operações estão bem definidas e, em caso afirmativo, comente o resultado obtido.

```
a. >> x + y                 b. >> x + A
c. >> x' + y'               d. >> A - [x ; y]
e. >> [x ; y']              f. >> [x ; y]
g. >> [A B]                 h. >> [A ; B]
i. >> A - 3                 j. >> A + B
k. >> A * B                 l. >> B * A
m. >> A .* B                n. >> B * A'
```

Exercício 6. Construa, de uma forma simples, os seguintes vectores:

$$u = (1,1,1,1,1,1,1,1,1,1); \qquad v = (1,2,3,4,5,6,7,8,9,10);$$
$$x = (0, 0.25, 0.5, 0.75, 1); \qquad y = (2,4,6,8,10,12,14,16);$$
$$z = (10,8,6,4,2,0,-2,-4,-6); \qquad w = (0,0,0,0,0,1,1,1,1,1).$$

Exercício 7. Usando os vectores definidos na questão anterior, construa:

$$U = (3,3,3,3,3,3,3,3,3,3); \qquad V = (2,4,8,16,32,64,128,256,512,1024);$$
$$X = (5, 5.25, 5.5, 5.75, 6); \qquad Y = (4,16,36,64,100,144,196,256);$$
$$Z = (-6,-4,-2,0,2,4,6,8,10); \qquad W = (0,0,0,1,1,1).$$

Exercício 8. Defina, no MATLAB, o vector coluna x constituído:

a) pelos números pares com início em 4 e término em 100;

b) por uma sequência decrescente de números inteiros com início em 5 e término em -5;

c) pelos números múltiplos de 3 compreendidos entre 10 e 132, dispostos por ordem decrescente;

d) por 100 números aleatórios, usando a função **rand**;

e) por 128 elementos com a seguinte forma (0 1 0 -1 0 1 ... 0 -1)$^{\mathsf{T}}$.

Exercício 9. Defina a matriz `A=magic(4)` e indique os comandos para:

a) construir uma matriz `B` cujas colunas são as colunas *pares* da matriz `A`;

b) construir uma matriz `C` cujas linhas são as linhas *ímpares* da matriz `A`;

c) converter `A` numa matriz 2×8;

d) calcular o inverso de cada elemento da matriz `A`;

e) calcular a inversa da matriz `A`;

f) calcular o quadrado de cada elemento da matriz `A`.

Exercício 10. Dado `x = [1 5 2 8 9 0 1]` e `y = [5 2 2 6 0 0 2]` execute e explique o resultado dos seguintes comandos:

```
a. >> x > y                    b. >> y < x
c. >> x == y                   d. >> x <= y
e. >> y >= x                   f. >> x | y
g. >> x & y                    h. >> x & (~y)
i. >> (x > y) | (y < x)        j. >> (x > y) & (y < x)
```

Exercício 11. Dado `x = 1:10` e `y = [3 1 5 6 8 2 9 4 7 0]` execute e interprete o resultado dos seguintes comandos:

```
a. >> x(x > 5)                      b. >> y(x <= 4)
c. >> x( (x < 2) | (x >= 8) )       d. >> y( (x < 2) | (x >= 8) )
e. >> y((x > 2) & (x < 8))          f. >> x(y < 0)
```

Exercício 12. Defina o vector `x=[3 15 9 12 -1 0 -12 9 6 1]` e indique o(s) comando(s) para:

a) atribuir o valor zero aos elementos de `x` que são positivos;

b) atribuir o valor 3 aos elementos de `x` que são múltiplos de 3 (use a função **rem**);

c) multiplicar os elementos pares de `x` por 5;

d) criar um vector `y` extraindo os valores de `x` que são maiores que 10;

e) atribuir o valor zero aos elementos de `x` que são menores do que a média;

f) atribuir aos elementos de `x` superiores à média, a sua diferença em relação à média.

Exercício 13. Construa uma tabela com os valores das funções sen(πx), cos(πx) e tan(πx), para $x = 0 : 0.25 : 2$, usando a função **linspace**.

Exercício 14. Determine a soma dos primeiros 100 números primos.

Exercício 15. Construa uma matriz aleatória 5×5 e determine:

a) o maior elemento da matriz; o valor máximo em cada coluna e em cada linha;

b) o índice de linha e coluna de todos os elementos que são superiores a 0.25.

Exercício 16. Construa um vector `idades` com as idades dos alunos da turma. Calcule a média, mediana, moda e desvio padrão dessas idades.

Exercício 17. Os comandos `>> x=1:10;class(x)`, produzem `ans= double`. Diga como poderia criar um vector com os 10 primeiros inteiros positivos.

Exercício 18. Use a função **pascal** para definir uma matriz de Pascal A de ordem 6.

a) Quantos números primos tem A?

b) Qual é o índice de linha e coluna desses primos?

Exercício 19.

a) Utilize o comando `help polyfun` para obter uma lista de funções para operar com polinómios.

b) Defina os polinómios $p(x) = 2x^3 - 3x^2 - 1$ e $q(x) = x^3 + 1$ e determine:

(i) $p * q$; p'; $p(5)$; $[p(1), p(2), p(3)]$;

(ii) as raízes de p e de q.

Exercício 20. Use as funções **bin2dec**, **dec2bin**, **base2dec** ou **dec2base** para:

a) obter a representação nas bases decimal, octal e hexadecimal, dos seguintes números representados na base binária: $(1011011)_2$ e $(1111111110000)_2$;

b) obter a representação nas bases binária, octal e hexadecimal dos seguintes números representados no sistema decimal: 1325 e 128.

Exercício 21.

a) Obtenha informação sobre as constantes do MATLAB **realmax**, **realmin** e **eps** e determine o seu valor.

b) Tendo em atenção que está a trabalhar num sistema de norma IEEE 754 em formato duplo, justifique os valores de **realmax** e **realmin** obtidos.

c) Qual a relação entre **eps** e a unidade de erro de arredondamento do sistema em que está a trabalhar?

Algoritmos, fluxogramas e pseudo-código

Conteúdo

1. Algoritmos

A palavra algoritmo deriva do nome do matemático persa Al-Khwarizmi (780-850). Este matemático e astrónomo introduz na sua obra "Aritmética" (em latim *Algoritmi de numero Indorum*) o sistema de numeração actual (indo-arábico). No seu texto, Al-Khwarizmi apresenta 9 símbolos indianos para representar os algarismos e um círculo para representar o zero. Depois explica como representar um número no sistema decimal posicional e descreve as operações de cálculo.

Ainda que os algoritmos datem de tempos babilónicos e os gregos tenham concebido algoritmos ainda hoje famosos (por exemplo, o algoritmo de Euclides para calcular o máximo divisor comum de dois números), foi Al-Khwarizmi o primeiro a conceber algoritmos tendo em conta a sua eficiência, para o caso concreto da determinação das raízes de equações. No seu tratado "Álgebra" é apresentada uma introdução compacta ao cálculo, usando regras para completar e reduzir equações. A palavra álgebra deriva do título desse livro *Hisab al-jabr w'al-muqabala*- álgebra é a tradução latina de *al-jabr*.

Definição: Um algoritmo é uma sequência ordenada, finita e não ambígua de instruções bem definidas para realizar uma determinada tarefa.

Exemplos: Receita culinária; montagem de um kit; ordenação de um conjunto.

Características de um algoritmo

Um algoritmo deve descrever exactamente como realizar determinada tarefa e deve ser:

1. completo;
2. preciso;
3. finito.

Representação de algoritmos

1. Linguagem natural/narrativa

   Os algoritmos são escritos em linguagem natural, por exemplo, o português. É o caso de uma receita culinária.

   ↓ muito "palavroso";

   ↓ sensível ao contexto - depende da experiência do leitor.

2. Fluxogramas ou Diagramas de Fluxo

   Representação gráfica que usa formas geométricas padronizadas para indicar as diversas acções e decisões que devem ser executadas para resolver o problema dado.

   ↓ pode tornar-se muito complexo;

   ↓ difíceis de alterar/modificar;

   ↓ detalhes técnicos podem sobrepor-se ao essencial

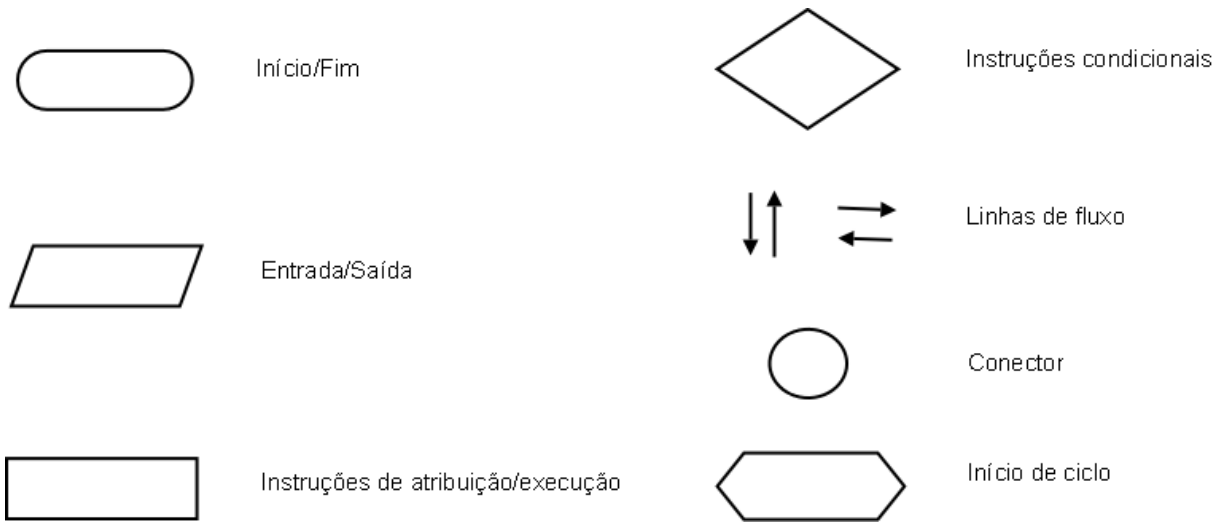
3. Pseudo-código

   Emprega uma linguagem intermédia entre a linguagem natural e uma linguagem de programação para descrever os algoritmos.

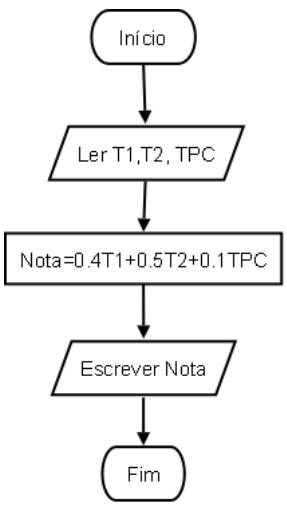
4. Linguagem de programação

2. Fluxogramas

Os fluxogramas descrevem o fluxo de um algoritmo através de um conjunto de símbolos standard. Os mais frequentes são:



Exemplo 1: Calcular a classificação final da avaliação periódica de Matemática Computacional I.



3. Pseudo-código

Pseudo-código é uma lista ordenada/numerada de instruções para realizar uma tarefa, escrita numa linguagem informal, mas mais próxima da linguagem de programação.

Exemplo 1:

(1.) Ler T1,T2 e TPC

(2.) Calcular Nota através de 0.4T1+0.5T2+0.1TPC

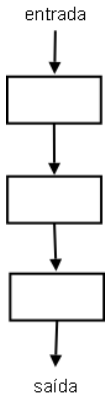
(3.) Escrever Nota

4. Estruturas lógicas de programação

A elaboração de um algoritmo pode envolver 3 estruturas lógicas fundamentais no controle do fluxo de dados e instruções. Estas 3 estruturas de controle são:

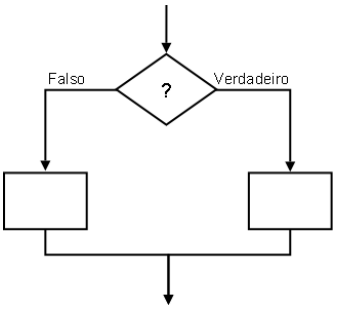
1. **Sequência**

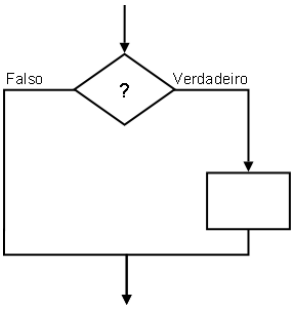
   As instruções são executadas uma após a outra, respeitando sempre a estrutura linear "de cima para baixo".

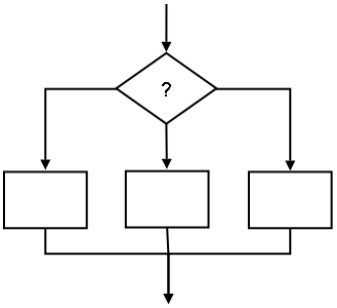


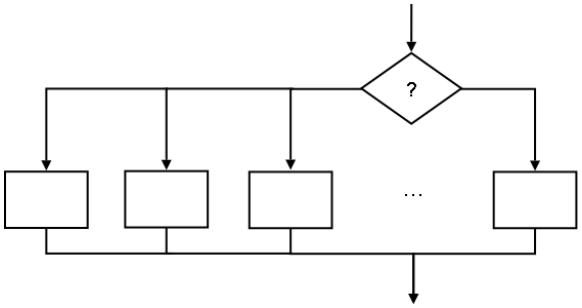
2. **Selecção**

   Esta estrutura exerce o controle sobre uma sequência de instruções a serem executadas, por meio de um teste ou verificação lógica



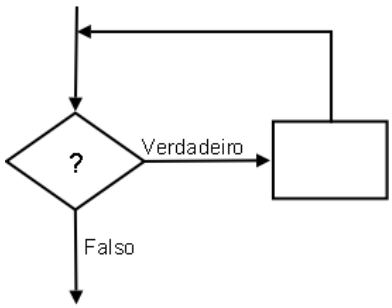




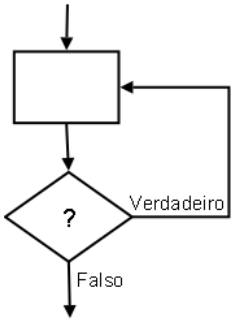


3. **Repetição**

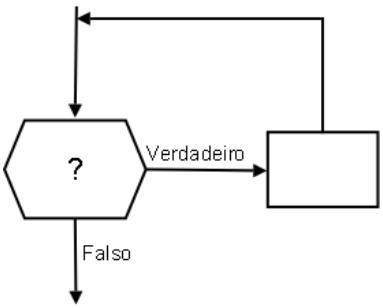
   Através de um teste ou verificação lógica, uma instrução ou uma conjunto de instruções é executado repetidamente.



A acção é executada repetidamente, enquanto o resultado for verdadeiro.
O teste precede a acção



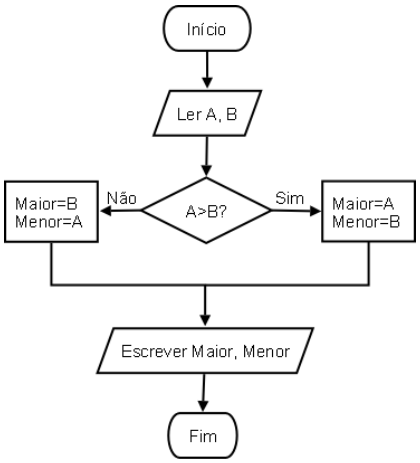
A acção é executada repetidamente, enquanto o resultado for verdadeiro.
A acção precede o teste.



À partida é indicado o número de vezes que a acção é repetida.
O teste precede a acção.

Exemplo 2: Ler dois números e escrevê-los por ordem decrescente.

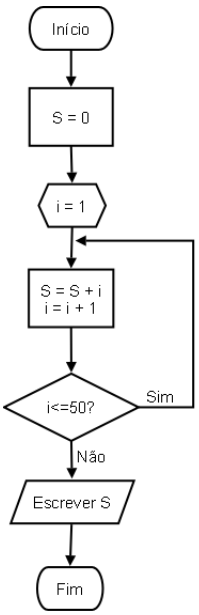
Fluxograma



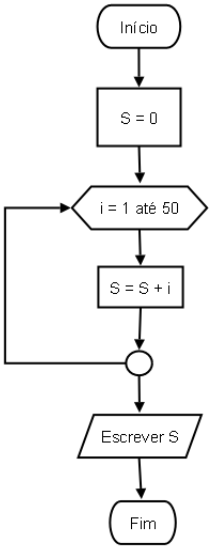
Pseudo-código

Ler A,B
Se A<B
    Maior=B
    Menor=A
Senão
    Maior=A
    Menor=B
Escrever Maior, Menor

Exemplo 3: Calcular $S = \sum_{i=1}^{50} i$



ou



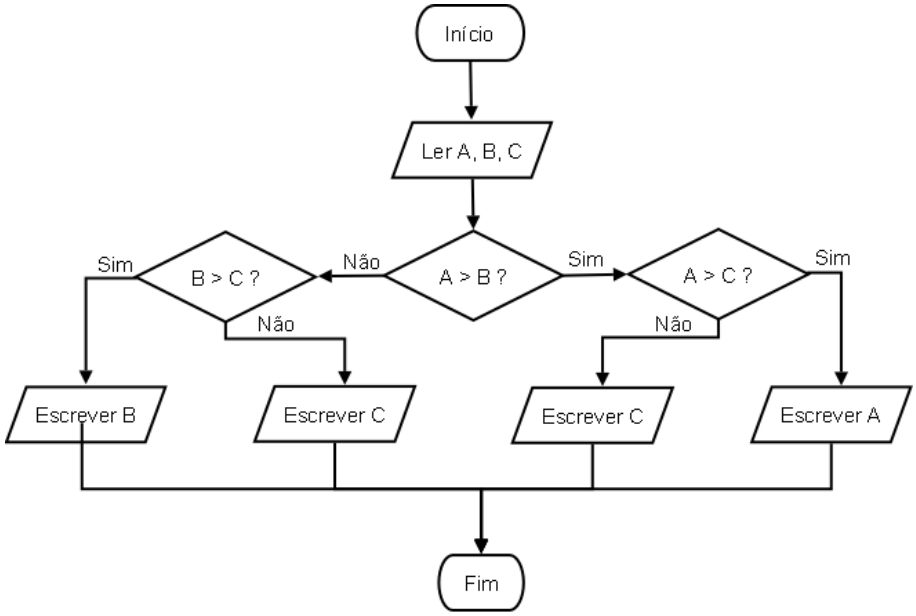
S=0
i=1
Repetir
    S=S+i
    i=i+1
enquanto i $\leq$ 50
Escrever S

ou

S=0
Para i=1 **até** 50
    S=S+i
Escrever S

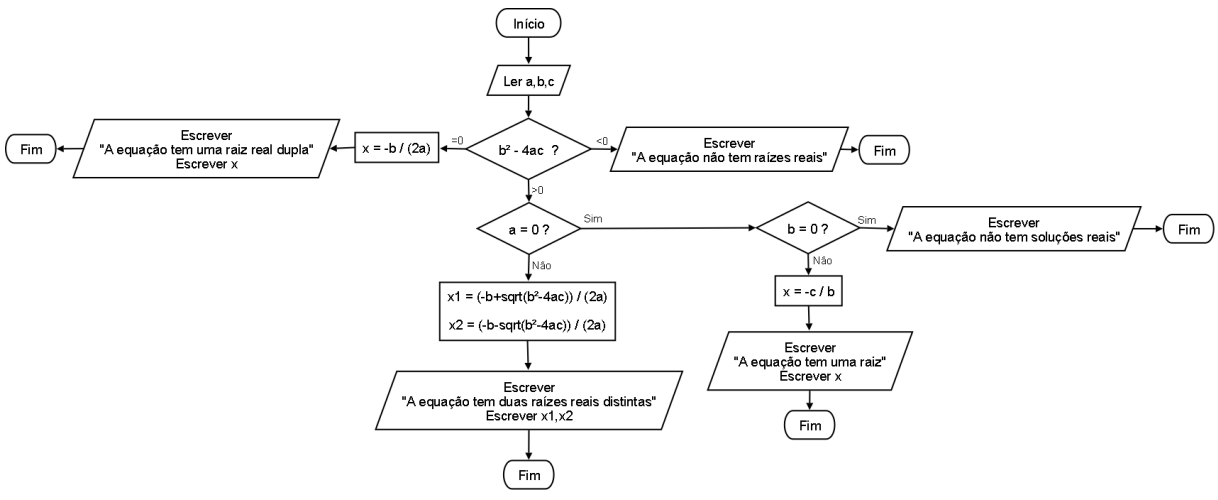
Obs: Como se pode escrever o algoritmo com o teste a preceder a acção - *teste à cabeça*?

Exemplo 4: Dados três números A, B e C, determinar o maior.




```
Ler A,B, C
Se A>B
    Se A>C
        Escrever A
    Senão
        Escrever C
Senão, se B>C
    Escrever B
Senão
    Escrever C
```

Exemplo 5: Dados a, b e c, determinar as raízes reais da equação ax^2+bx+c=0.



```
Ler a,b,c
Se  b²-4ac < 0
    Escrever A equação não tem raízes reais.
Se b²-4ac = 0
    x=-b/(2a)
    Escrever A equação tem uma raiz real dupla.
    Escrever x
Se b²-4ac > 0
    Se a=0
        Se b=0
            Escrever A equação não tem raízes reais.
        Senão
            x=-c/b
            Escrever A equação tem uma raiz real.
            Escrever x
    Senão
        x1=(-b+sqrt(b²-4ac))/(2a)
        x2=(-b-sqrt(b²-4ac))/(2a)
        Escrever A equação tem duas raízes reais distintas.
        Escrever x1,x2
```

5. Exercícios

Exercício 1. Diga qual o objectivo dos seguintes algoritmos:

```
Ler a, b
Se a < b fazer
    x = a
    a = b
    b = x
Escrever a, b
```

```
f = 1
Para n = 1 até 12
    f = f × n
Escrever f
```

```
f = 1
n = 1
Enquanto f ≤ 12 fazer
    n = n + 1
    f = f × n
Escrever f
```

Exercício 2. Escreva um algoritmo para ler dois números e os escrever por ordem decrescente.

Exercício 3. Escreva um algoritmo para calcular $S = \sum_{i=1}^{50} i$.

Exercício 4. Escreva um algoritmo para, dado n, calcular $P = \prod_{i=1}^{n} i$. O que é P?

Exercício 5. Dados três números A, B e C, escreva um algoritmo para determinar o maior.

Exercício 6. Considere o seguinte algoritmo em pseudo-código:

Ler um número natural N
i=1
Enquanto $i \times i < N$ **fazer**
    $i = i + 1$
Se $i \times i = N$ **então**
    Escrever "Sim"
Senão
    Escrever "Não"

a) Qual o resultado do algoritmo anterior para $N = 15,\ 16,\ 24,\ 25$?

b) Que propriedade de N está este algoritmo a testar?

Exercício 7. Relembre que um número natural n é **triangular** se $n = 1 + 2 + \cdots + k$, para algum $k \in \mathbb{N}$.

Exemplo: Primeiros 4 números triangulares: 1, 3, 6, 10.

O algoritmo seguinte pretendia verificar se um dado número n é triangular, mas tem um erro. Detecte-o e corrija-o.

Ler um número natural N
$i = 1$
$S = 0$
Enquanto $S < N$ **fazer**
    $i = i + 1$
    $S = S + i$
Se $S = N$ **então**
    Escrever "O número é triangular"
Senão
    Escrever "O número não é triangular"

Exercício 8. Qual o objectivo do seguinte algoritmo?

Ler um número natural N
$M = 0$
Enquanto $N \neq 0$ **fazer**
    $M = 10M + \text{mod}(N, 10)$
    $N = \text{int}(N/10)$
Escrever M

Exercício 9. Dados a, b e c, escreva um algoritmo para determinar as raízes reais da equação ax^2+bx+c=0.

Exercício 10. No calendário gregoriano, um ano é bissexto se é divisível por 400 ou se o ano é divisível por 4 mas não por 100. Escreva um algoritmo para determinar se um dado ano é bissexto.

Exercício 11. Escreva um algoritmo para determinar os divisores de um número.

Exercício 12. Escreva um algoritmo em pseudo-código para determinar os primeiros k números primos (pode usar a função **isprime** do Matlab).

Conteúdo

1. Ficheiros-M

As instruções em MATLAB são geralmente dadas e executadas linha a linha. É, no entanto, possível executar uma sequência de comandos que está guardada num ficheiro. Ficheiros que contêm código em linguagem MATLAB são chamados *ficheiros-M* (em inglês, *M-files*) e são do tipo **nome_ficheiro.m**.

Um ficheiro-M consiste numa sequência de comandos MATLAB que pode inclusivamente fazer referência a outros ficheiros-M. Os ficheiros-M podem criar-se usando o editor de texto associado ao MATLAB . Para aceder a este editor basta seleccionar a opção **New** seguida de **M-file** do menu **File**, para criar um novo ficheiro-M, ou **Open** para editar um ficheiro-M já existente. Qualquer texto a seguir ao símbolo % é considerado comentário.

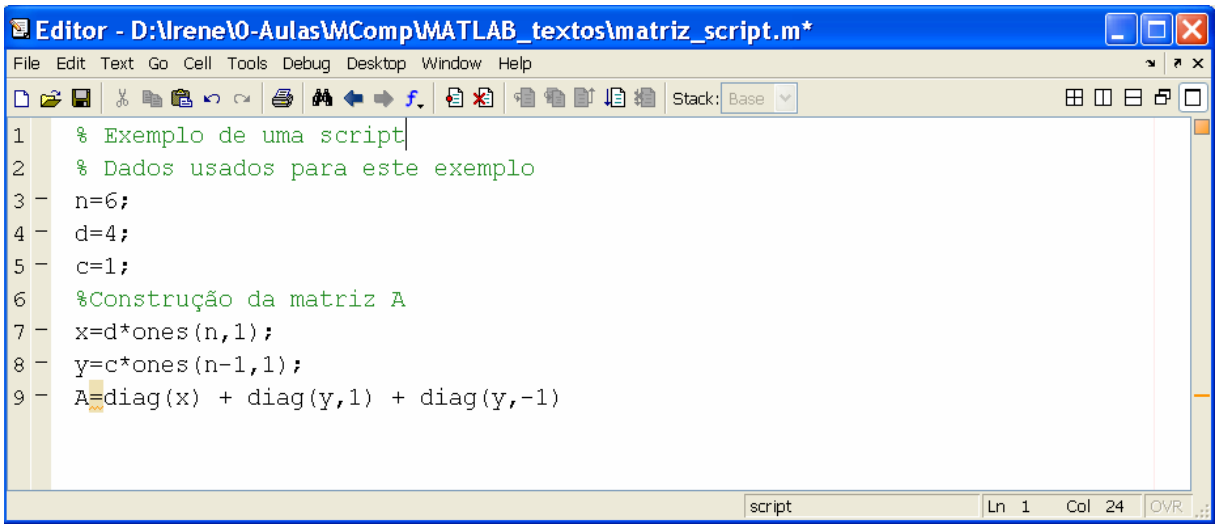
Há dois tipos de ficheiros-M que podem ser usados: *scripts* e funções.

Scripts

Quando uma *script* é invocada, o MATLAB executa os comandos encontrados no ficheiro e as variáveis que aparecem na *script* podem ser de novo usadas. Este tipo de ficheiro é muito útil quando há necessidade de executar um grande número de operações.

Suponhamos, por exemplo, que se pretende gerar uma matriz tridiagonal de ordem $n \times n$ que tem o valor d ao longo da diagonal principal e o valor c nas diagonais abaixo e acima da diagonal principal.

Para o efeito, no menu do MATLAB, seleccione a opção **New M-file** do menu **File**, para criar um ficheiro de texto chamado, por exemplo, *matriz_script.m* com a seguinte sequência de instruções



Para executar esta *script* basta fazer, na janela do MATLAB

```
>> matriz_script
```

obtendo-se

```
A =
     4     1     0     0     0     0
     1     4     1     0     0     0
     0     1     4     1     0     0
     0     0     1     4     1     0
     0     0     0     1     4     1
     0     0     0     0     1     4
```

A matriz A e os vectores x e y estão disponíveis e podem ser de novo utilizados; por exemplo, se fizer

```
>> sum(x)-sum(y)
```

obterá

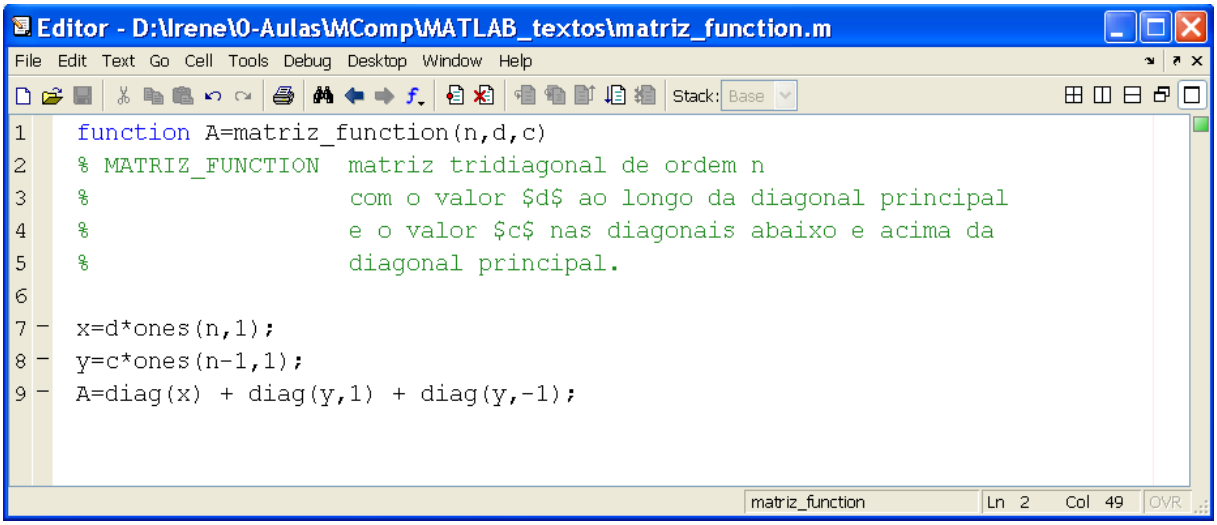
```
ans =
    19
```


Um ficheiro-M que contém a palavra **function** no início da primeira linha é chamado uma **função**. As funções diferem das *scripts* na medida em que podem ser usados argumentos e as variáveis definidas e manipuladas dentro de uma função são locais, i.e. não operam globalmente no espaço de trabalho.

Uma função é definida do seguinte modo:

function *parâmetros_saida=nome_função(parâmetros_entrada)*

Para resolver o problema da construção de uma matriz de ordem n da forma atrás referida pode criar-se uma função *matriz_function.m* do género



A função **matriz_function** passa a fazer parte das funções do MATLAB. As primeiras linhas de comentário que aparecem na função podem ser acedidas via *help*.

```
>> help matriz_function
  MATRIZ_FUNCTION  matriz tridiagonal de ordem n
                   com o valor $d$ ao longo da diagonal principal
                   e o valor $c$ nas diagonais abaixo e acima da
                   diagonal principal.
```

Para construir a matriz `A` definida anteriormente basta fazer

```
>> matriz_function(6,4,1)

ans =
     4     1     0     0     0     0
     1     4     1     0     0     0
     0     1     4     1     0     0
     0     0     1     4     1     0
     0     0     0     1     4     1
     0     0     0     0     1     4
```

Neste caso as variáveis x e y não estão definidas no espaço de trabalho.

```
>> x
??? Undefined function or variable 'x'.

>> y
??? Undefined function or variable 'y'.
```

O caso de funções com mais de um parâmetro de entrada ou saída é tratado de modo análogo. A função seguinte calcula as raízes de um polinómio do segundo grau.

```
function [x1,x2]=raizes(a,b,c)
% RAIZES Calcula as raizes  da equacao ax^2+bx+c=0,
d=sqrt(b*b-4*a*c);
x1=(-b+d)/(2*a);
x2=(-b-d)/(2*a);
```

Listagem 1. Função com dois parâmetros

Para resolver, por exemplo, a equação $x^2 + x + 2 = 0$ basta fazer

```
>> [r1,r2]=raizes(1,1,2)
```

obtendo-se, então:

```
r1 =
 -0.5000 + 1.3229i

r2 =
 -0.5000 - 1.3229i
```

2. Instruções de *Input* e *Output*

É possível introduzir um texto ou um valor numérico através do teclado, de uma forma interactiva. Para tal, pode usar-se a função **input** cuja forma é:

variavel_numerica=**input***('texto')*

ou

variavel_string=**input***('texto','s').*

No primeiro caso, aparece no ecrã o texto *texto* e o utilizador deve introduzir um valor numérico que será atribuído à variável *variavel_numerica*. O segundo caso é usado quando se pretende introduzir um valor não numérico.

Como exemplo da utilização desta função, considerem-se as instruções

```
>> vn=input (' introduza o valor de vn')
```

e

```
>> vs=input (' Pretende continuar? (SIM/NAO)','s')
```

Se for introduzido o valor 10 no primeiro caso e SIM no segundo, então as instruções anteriores originam $vn = 10$ e $vs =$ SIM.

A função **disp** permite escrever um texto ou valor no ecrã. Assim, o comando

```
>> vs=disp(A)
```

escreve a matriz A no ecrã.

Para escrever uma frase no ecrã deve rodear-se essa frase do símbolo '. Por exemplo,

```
>> disp(' Estou a escrever esta frase' )
```

Quando se pretender escrever um conjunto de várias frases, devem separar-se as frases por vírgulas e rodear o conjunto com os símbolos [e]. Por exemplo,

```
>> disp([' Estou a escrever esta frase ',' ha ', ' minutos' ] )
```

Para combinar texto e valores numéricos devem converter-se estes últimos a "texto" através da função **num2str**. O comando

```
>> disp([' Estou a escrever esta frase ',' ha ',num2str(n), ' minutos' ] )
```

escreverá no ecrã, no caso $n = 10$, a frase

```
 Estou a escrever esta frase  ha 10 minutos
```

Para outros formatos de entrada/saída, invoque o **help** do MATLAB para conhecer melhor o uso das instruções **fprintf**, **sprintf**, **fwrite**, **fscanf**, etc.

3. Instruções de controle

Como já foi referido, normalmente as instruções em MATLAB são executadas sequencialmente, isto é, depois de uma instrução ter sido executada, é executada a instrução imediatamente a seguir. As seguintes transferências de controle podem ser usadas para alterar a sequência de execução das instruções de uma *script* ou função: ciclos **for** e **while**, instruções **if** e **switch** e ainda **break**, **error** e **return**. Estas instruções são semelhantes às encontradas na maioria das linguagens de programação.

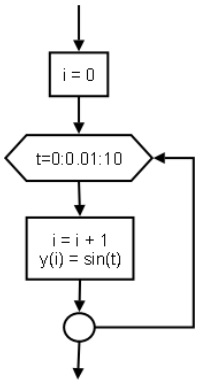
Ciclos **for**

A instrução **for** permite que um grupo de instruções seja executado repetidas vezes. Tem a forma

for *variavel=expressão*
instruções
end

onde *expressão* é usualmente da forma $\mathbf{m}:\mathbf{n}$ ou $\mathbf{m}:\mathbf{i}:\mathbf{n}$, sendo $\mathbf{m}$ o valor inicial, $\mathbf{i}$ o valor incremental e $\mathbf{n}$ o valor final da *variavel*.

Suponhamos que pretendemos calcular um vector com o valor da função seno em 1001 pontos igualmente espaçados do intervalo $[0, 10]$. A maneira mais óbvia de obter este vector é:




```
i=0
for t=0:.01:10
    i=i+1;
    y(i)=sin(t);
end
```

Listagem 2. Ciclos **for**

Note-se que `t=0:.01:10` é apenas um vector linha. De facto, qualquer vector linha pode ser usado num ciclo **for**. Por exemplo,

```
x=1:100;soma=0;
for j=find(isprime(x))
    soma=soma+x(j);
end
```

Listagem 3. Ciclos **for** - outro exemplo

calcula a soma de todos os números primos inferiores a 100.

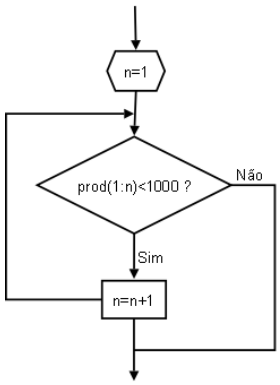
Ciclos **while**

A instrução **while** permite que um grupo de instruções seja executado um número indefinido de vezes, enquanto uma condição for satisfeita. Tem a forma

while *expressão*
    instruções
end

onde *expressão* é uma expressão de relação da forma *e1Re2*, sendo *e1* e *e2* expressões aritméticas e *R* um dos operadores de relação já definidos nas notas *Iniciação ao Matlab* (ver pag. 16).

O exemplo seguinte ilustra o uso de um ciclo **while**, onde se pretende calcular o primeiro inteiro n para o qual $n!$ é um número com 4 dígitos.




```
n=1;
while prod(1:n)< 1000
  n=n+1;
end
n
```

Listagem 4. Ciclos **while**

Nota: O tempo de execução de programas em MATLAB pode ser substancialmente reduzido, se se tiver a preocupação de "vectorizar" os algoritmos. Vectorizar significa converter ciclos em operações com vectores ou matrizes. Por exemplo as instruções,

```
>> t=0:.1:10;
>> y=sin(t);
```

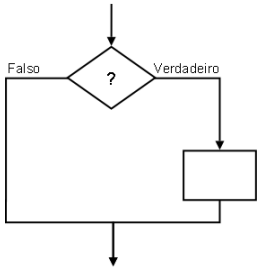
correspondem a uma versão vectorizada do exemplo ilustrativo do ciclo **for** considerado anteriormente.

Instruções **if – elseif – else**

Os blocos **if** permitem alterar a ordem de execução de uma sequência de instruções, se determinada condição for (ou não) satisfeita.

➠ **if · · · end**

if *expressão_logica*
  instruções
  avaliadas, se expressão_logica verdadeira
end

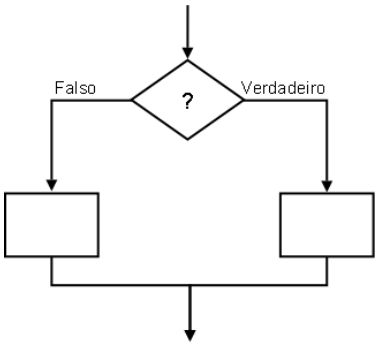


```
if a>0
   b=a;
   disp('a e positivo');
end
```

Listagem 5. Uso de **if – end**

➠ **if · · · else · · · end**

if *expressão_logica*
instruções
avaliadas, se expressão_logica verdadeira
else
instruções
avaliadas, se expressão_logica falsa
end

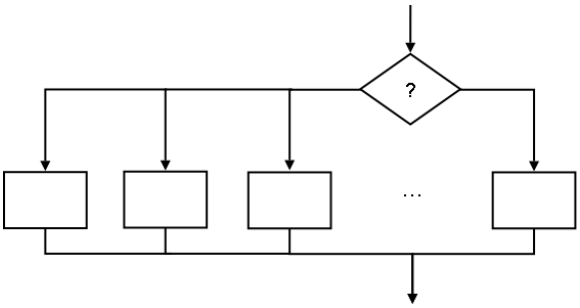



```
if T>37
   febre=1;
   disp('Com febre');
else
   febre=0;
   disp('Sem febre');
end
```

Listagem 6. Uso de **if – else – end**

➠ **if · · · elseif · · · else · · · end**

if *expressão_logica1*
instruções
avaliadas, se expressão_logica1 verdadeira
elseif *expressão_logica2*
instruções
avaliadas, se expressão_logica2 verdadeira
elseif *expressão_logica3*
instruções
avaliadas, se expressão_logica3 verdadeira
...
else
instruções
avaliadas, se nenhuma das expressões anteriores é verdadeira
end



Cada uma das expressões *expressao_logica1*, *expressao_logica2*, etc é uma expressão de relação da forma *e1Re2*, sendo *e1* e *e2* expressões aritméticas e *R* um dos operadores de relação definidos anteriormente.

```
if altura>190
   disp('Muito alto');
elseif altura>170
   disp('Alto');
elseif altura<150
   disp('Baixo');
else
   disp('Mediano');
end
```

Listagem 7. Uso de **if – elseif – else – end**

Podem também combinar-se estes operadores de relação com os operadores lógicos ou ainda usar-se funções lógicas.

Instruções **switch** e **case**

A instrução **switch** executa um grupo de instruções, dependendo do valor de uma variável ou expressão. Tem a forma

switch *expressao*
  case *caso1*
    instruções
  case {*caso2a,caso2b, ...*}
    instruções
      ...
  otherwise
    instruções
end

Se $n = 23$, qual será o resultado da seguinte *script*?

```
switch rem(n,4)
case 0
  a=ones(n)
case {1,2}
  a=eye(n)
otherwise
  a=zeros(n)
end
```

Listagem 8. Uso de **switch – case – otherwise**

Além das instruções de controle já definidas, o MATLAB dispõe de quatro comandos - **break**, **continue**, **error** e **return**, para controlar a execução de *scripts* e funções. Uma breve descrição destas instruções é feita de seguida.

A instrução **break** permite sair de um ciclo **for** ou **while**. A instrução **continue** pode ser usada em ciclos **for** ou **while**. Quando o MATLAB encontra a instrução **continue** dentro de um ciclo, passa imediatamente para a instrução final desse ciclo, ignorando todas as instruções entre **continue** e a declaração **end**.

Um exemplo trivial da utilização destas instruções encontra-se na *script* apresentada na Listagem 9. Este ficheiro-M escreve, apenas, no ecrã os números entre 5 e 10.

```
for i=1:20
   if i<5
      continue
   elseif i>10
      break
   end
   disp(i);
end
```

Listagem 9. Uso de **break** e **continue**

O comando

error ('*mensagem_de_erro*')

dentro de uma função ou *script*, aborta a execução, exibe a mensagem de erro *mensagem_de_erro* e faz regressar o controle ao teclado.

O comando **return** faz regressar o controle à função invocadora ou ao teclado.

```
function [x1,x2]=zeros_p(p)
if length(p) ~= 3
   error('p deve ser um polinomio de grau 2');
end
delta=p(2)*p(2)-4*p(1)*p(3);
if delta<0
   disp('Este polinomio nao tem zeros reais');
   return
else
   x1=(-p(2)+sqrt(delta))/(2*p(1));
   x2=(-p(2)-sqrt(delta))/(2*p(1));
end
disp('Este polinomio tem zeros reais');
```

Listagem 10. Uso de **error** e **return**

A Listagem 10 ilustra o uso das instruções **error** e **return**. Neste caso, pretende-se apenas obter os zeros reais de um polinómio do segundo grau, cujos coeficientes devem ser indicados num vector.

4. Funções de novo!

Variáveis locais e globais

Quando uma variável é definida na janela de comandos do MATLAB, esta fica automaticamente gravada no espaço de trabalho *MATLAB Workspace*. O mesmo se passa se a variável for definida numa *script*, uma vez que uma *script* é apenas um conjunto de comandos. Tal não acontece com as funções. Na verdade, as funções não partilham o mesmo espaço de trabalho. Isto significa que as variáveis definidas numa função são variáveis **locais** ao espaço de trabalho da função. Cada função tem o seu próprio espaço de trabalho temporário, o qual é criado a cada chamada e apagado quando a função completa a execução.

Ocasionalmente, pode ser necessário definir variáveis que existam em mais que um espaço de trabalho, incluindo eventualmente o próprio *MATLAB Workspace*. Estas variáveis devem ser então declaradas como **globais**. Para isso, todos os "locais" que precisem de partilhar essa variável (funções, *scripts* ou janela de comandos) devem ter uma linha inicial que identifique as variáveis em causa como globais, usando o comando **global**. Por exemplo, o uso da instrução

```
>> global VARIAVEL_GLOBAL
```

permite a partilha da variável VARIAVEL_GLOBAL.

As variáveis globais podem ser apagadas fazendo

```
>> clear global
```

A função **isglobal** pode ser usada para verificar se uma dada variável é ou não global. Uma lista completa de todas as variáveis globais pode ser acedida via

```
>> who global
```

Na prática de programação, o uso de variáveis globais não é encorajado. Todavia se estas forem usadas, aconselha-se que o nome destas variáveis seja longo e escrito em letra maiúscula (para evitar possíveis conflitos com outras variáveis globais).

Subfunções

Um ficheiro-M pode conter código para definir uma ou mais funções. A primeira função que aparece e que tem o mesmo nome do ficheiro-M é chamada *função principal*. As restantes funções são chamadas *subfunções* e são visíveis apenas para a função principal e as outras subfunções do ficheiro-M.

As subfunções são definidas de modo análogo às funções e podem aparecer por qualquer ordem no ficheiro-M que as contém, desde que a função principal seja a primeira.

Geralmente, as subfunções realizam tarefas que precisam de ser executadas separadamente da função principal, mas que, em princípio, terão pouco interesse fora do âmbito em que foram criadas. A técnica de usar subfunções permite evitar a proliferação de ficheiros-M.

```
function [media,mediana]=exemplo_subfuncoes(x)
% EXEMPLO_SUBFUNCOES calcula a media e a mediana de uma amostra
%
%  Este exemplo permite ilustrar o uso de subfuncoes
%
n=length(x);
media=mean(x,n);
mediana=median(x,n);

function y=mean(x,n)
% MEAN calcula a media de um conjunto de valores
%      Nao ha conflito com a funcao interna MEAN do MATLAB
%      porque esta funcao e executada primeiro.
disp('Funcao invocada:subfuncao mean da funcao exemplo_subfuncoes')
y=sum(x)/n;

function y=median(x,n)
% MEDIAN calcula a mediana de um conjunto de valores
%        Nao ha conflito com a funcao interna MEDIAN do MATLAB
disp('Funcao invocada:subfuncao median da funcao exemplo_subfuncoes')
xordenado=sort(x);
if rem(n,2)==1
    % Se n e impar, a mediana e o elemento na posicao (n+1)/2
    y=xordenado((n+1)/2);
else
    % Se n par, a mediana e a media entre os elementos
    % nas posicoes n/2 e n/2+1
    y=(xordenado(n/2)+xordenado(n/2+1))/2;
end
```

Listagem 11. Funções e subfunções

No exemplo anterior, foi criado um ficheiro-M chamado *exemplo_subfuncoes.m* que define três funções: a função principal, chamada **exemplo_subfuncoes** e as duas subfunções **mean** e **median**. Usamos propositadamente **mean** e **median** como nomes para as subfunções, os quais são os nomes de duas funções internas do MATLAB. Não há qualquer conflito neste caso, uma vez que quando estas funções são invocadas no ficheiro-M *exemplo_subfuncoes.m*, o MATLAB verifica primeiro se há alguma subfunção desta função com esses nomes e, em caso afirmativo, são estas as funções avaliadas. De facto, fazendo

```
>> x=[1 5 2 -3 8 9];
>> [minha_media,minha_mediana]=exemplo_subfuncoes(x)
>> outra_media=mean(x)
>> outra_mediana=median(x)
```

obtém-se

```
    Funcao invocada:subfuncao mean da funcao exemplo_subfuncoes
    Funcao invocada:subfuncao median da funcao exemplo_subfuncoes
minha_media =
    3.6667
minha_mediana =
```

```
   3.5000
outra_media =
    3.6667
outra_mediana =
    3.5000
```

O comando **help** permite ter acesso à ajuda da função principal. Pode também aceder-se às ajudas das subfunções, indicando o nome da função principal e da subfunção, como a seguir se exemplifica.

```
>> help exemplo_subfuncoes

  EXEMPLO_SUBFUNCOES calcula a media e a mediana de uma amostra
           Este exemplo permite ilustrar o uso de subfuncoes

>> help exemplo_subfuncoes/mean

  MEAN calcula a media de um conjunto de valores
       Nao ha conflito com a funcao interna MEAN do MATLAB
       porque esta funcao e executada primeiro.
```

Funções anónimas

É possível definir funções, de forma simples, através de uma linha de comando, sem recorrer à criação (e consequente armazenamento em disco) de um ficheiro-M. Esta alternativa passa pela utilização das chamadas *funções anónimas*. A sintaxe para criar uma função anónima é a seguinte:

$$f\_anonima = @(argumentos)\ expressão$$

Um exemplo muito simples de uma função anónima é:

```
>>g=@(x) x^2+2*x+3
```

Para avaliar esta função basta fazer

```
>> g(2)
ans =
    11
```

Nota: É boa prática, definir sempre uma função anónima de forma que possa operar sobre escalares, vectores ou matrizes. Se

```
>> f=@(x) x.^2+2*x+3
```

qual será o valor de `g(1:5)` e `f(1:5)`?

As funções anónimas podem também ter mais que um argumento. Por exemplo, a função $h(x, y) = (x^2 + y^2, x + y)$ poderia ser definida e avaliada em `x=2` e `y=4` da seguinte forma

```
>> h=@(x,y) [x.^2+y.^2 x+y]
>> h(2,4)
```

Outra vantagem do uso de funções anónimas diz respeito à possibilidade de usar implicitamente variáveis definidas no espaço de trabalho, sem ter que as declarar como globais. Por exemplo:

```
>> peso1=0.4;peso2=0.5;peso3=0.1;
>> notafinal=@(T1,T2,TPC) peso1*T1+peso2*T2+peso3*TPC;
>> notafinal(8,11,14)
ans =
   10.1000
```

Quando uma função é um argumento

Muitas funções em MATLAB têm como argumentos outras funções. O MATLAB chama genericamente a estas funções **function functions** (faça `help funfun` para obter uma lista completa deste tipo de funções). Existem várias formas de passar uma função como argumento, dependendo da forma como esta foi escrita.

Uma forma é usar funções anónimas. Por exemplo, a função **fzero** permite encontrar um zero de uma função de uma variável. Podemos encontrar um zero de $f(x) = \sin(x)$ próximo de 3, fazendo

```
>> f=@(x) sin(x);
>> fzero(f,3)
ans =
    3.1416
```

É também possível passar a função `sin` como argumento, usando a sintaxe especial

```
>> fzero(@sin,3)
ans =
    3.1416
```

O nome `@sin` é chamado *function_handle* e é uma forma de referir abstractamente uma função, em vez de a invocar directamente. *Function_handle* é um tipo de dado do MATLAB que contém toda a informação necessária para avaliar uma função. Uma função tipo *function_handle* pode também ser criada colocando o caracter **@** atrás do nome da função definida anteriormente através de um ficheiro-M.

Para obter informação sobre uma *function_handles* podem usar-se as funções **fun2str** ou **functions**.

```
>> func2str(f)

ans =

@(x)sin(x)

>> functions(g)
```

```
ans =

     function: '@(x)x^2+2*x+3'
         type: 'anonymous'
         file: ''
    workspace: {[1x1 struct]}
```

Do ponto de vista da eficiência e versatilidade, o uso de funções do tipo *function_handles* deve ser encorajado, especialmente se estas forem passadas como argumento para outras funções. Mais informações acerca deste tipo de funções, podem ser obtidas no Capítulo *Function Handles* do manual do MATLAB .

Funções recursivas

As funções em MATLAB podem ser recursivas, isto é, podem chamar-se a si próprias. A recursividade é, de facto, uma ferramenta poderosa, mas nem sempre corresponde à melhor forma de programação.

Apresenta-se, a título de exemplo, a função **fibfun** que se encontra na Versão 6 do MATLAB. Esta função permite obter valores da sucessão

$$\begin{aligned} F_1 &= 1 \\ F_2 &= 1 \\ F_n &= F_{n-1} + F_{n-2},\ n > 2 \end{aligned}$$

i.e. determina recursivamente o n-ésimo número de Fibonacci.

```
function f = fibfun(n)
% FIBFUN For calculating Fibonacci numbers.
% Incidentally, the name Fibonacci comes from Filius Bonassi,
% or "son of Bonassus".
if n > 2
    f = fibfun(n - 1) + fibfun(n - 2);
else
    f = 1;
end;
```

Listagem 12. Função **fibfun** do MATLAB.

5. Gráficos - 2D

O MATLAB dispõe de um grande número de facilidades gráficas que podem ser usadas directamente ou acedidas a partir de funções ou *scripts*. Tendo em conta os objectivos específicos destes apontamentos, centraremos a nossa atenção em gráficos básicos 2-D. Todos os pormenores relativos às ferramentas de visualização incluídas nesta versão do MATLAB podem ser obtidos no guia *Using MATLAB Graphics* ou invocando o **help** para `graph2d, graph3d, specgraph` ou `graphics`.

O comando mais simples e, talvez o mais útil para produzir gráficos 2-D tem a forma

$$\mathbf{plot}(\textit{Abcissas,Ordenadas,'estilo'})$$

onde *Abcissas* e *Ordenadas* são vectores (com a mesma dimensão) que contêm as abcissas e ordenadas de pontos do gráfico e *estilo* é um argumento opcional que especifica o estilo da linha ou ponto a desenhar. Na tabela seguinte estão indicados alguns dos estilos que é possível definir.

cor		ponto		linha	
y	amarela	.	ponto(.)	$-$	contínua
m	magenta	**o**	círculo (o)	:	ponteada
c	cião	**x**	cruz(x)	$-.$	'traço-ponto'
r	vermelha	$+$	mais($+$)	$--$	tracejada
g	verde	$*$	estrela(*)		
b	azul	**s**	quadrado		
w	branca	**d**	losango($\diamondsuit$)		
k	preta	**v**	triângulo (∇)		

A função **plot** também permite o uso de apenas um vector como argumento. Se $Pontos = [x_1\ x_2 \cdots x_n]$, o comando

$$\mathbf{plot}(\textit{Pontos})$$

desenha os pontos de coordenadas $(i, x_i),\ i = 1, \cdots, n.$

Títulos, designação dos eixos, legendas e outras características, podem ser acrescentadas a um dado gráfico, usando as funções **title**, **xlabel**, **ylabel**, **grid**, **text** **legend**,etc. Estas funções têm a forma seguinte.

Designação	Descrição
title('*título*')	produz um *título* na parte superior do gráfico
xlabel('*nome_x*')	o eixo dos xx é designado por *nome_x*
ylabel('*nome_y*')	o eixo dos yy é designado por *nome_y*
grid	coloca uma quadrícula no gráfico
text(x,y,'*texto_em_x_y*')	escreve o texto *texto_em_x_y* na posição (x, y)
gtext('*texto*')	permite colocar *texto* numa posição a indicar com o rato
legend('*texto1*','*texto2*')	produz uma legenda com *texto1* e *texto2*
legend off	retira legenda

É também possível controlar os limites dos eixos através do comando **axis**. A função **axis** tem várias opções que permitem personalizar os limites, a escala, a orientação, etc, de um gráfico. Escrevendo

$$\mathbf{axis}([xmin\ \ xmax\ \ ymin\ \ ymax])$$

os limites passam a ser $xmin$ e $xmax$ para o eixo dos xx e $ymin$ e $ymax$ para o eixo dos yy. Este comando deve aparecer depois do comando **plot**. A função **axis** também aceita palavras chave

para controlar os eixos. Por exemplo, **axis square**, **axis equal**, **axis auto**, **axis on**, etc. (Faça **help axis**).

Numa mesma figura podem sobrepor-se vários gráficos, recorrendo ao comando **hold**. As instruções

hold on
plot (x_1, y_1)
plot (x_2, y_2)
plot (x_3, y_3)
hold off

originam a sobreposição de três gráficos. Este objectivo também pode ser atingido, usando

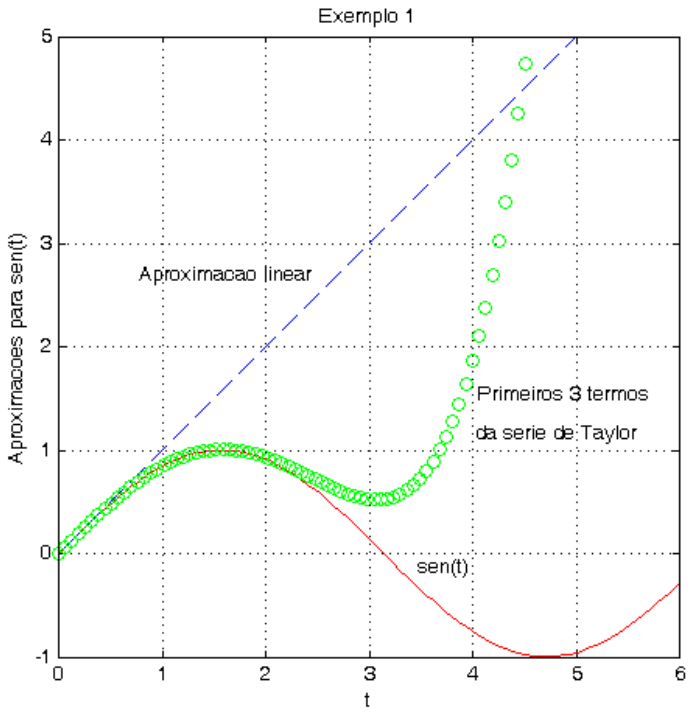
$$\textbf{plot}\ (x_1, y_1, x_2, y_2, x_3, y_3)$$

O comando **hold** é especialmente útil quando o conjunto de pontos a desenhar não está disponível todo ao mesmo tempo.

Na figura seguinte estão representados simultaneamente os gráficos das funções

$$f_1(t) = \text{sen}t, \quad f_2(t) = t, \quad f_3(t) = t - \frac{t^3}{3!} + \frac{t^5}{5!}.$$

Estes gráficos foram obtidos recorrendo à *script* apresentada na Listagem 13, onde foram usadas várias opções da instrução **plot**.



```
t=linspace(0,2*pi);
y1=sin(t);
y2=t;
y3=t-(t.^3)/6+(t.^5)/120;
plot(t,y1,'r',t,y2,'b--',t,y3,'go')
axis equal
axis([0 6 -1 5])
grid
xlabel('t')
ylabel('Aproximacoes para sen(t)')
title('Exemplo 1')
text(3.5,0,'sen(t)')
gtext('Aproximacao linear')
gtext('Primeiros 3 termos')
gtext('da serie de Taylor')
```

Listagem 13. Representação gráfica de funções: uso de **plot**

Alternativamente poder-se-ia obter o gráfico anterior, recorrendo à função **fplot**. Na listagem seguinte apresenta-se essa solução.

```
y1=@(t)sin(t);
y2=@(t)t;
y3=@(t) t-(t.^3)/6+(t.^5)/120;
hold on
fplot(y1,[0,2*pi],'r')
fplot(y2,[0,2*pi],'b--')
fplot(y3,[0,2*pi],'go')
hold off
axis equal
axis([0 6 -1 5])
grid
xlabel('t')
ylabel('Aproximacoes para sen(t)')
title('Exemplo 1')
text(3.5,0,'sen(t)')
gtext('Aproximacao linear')
gtext('Primeiros 3 termos')
gtext('da serie de Taylor')
```

Listagem 14. Representação gráfica de funções: uso de **fplot**

A versão 7 do MATLAB contém uma nova *interface* gráfica que permite criar e editar gráficos sem usar código. Use o menu **Help** para obter informações.

Exercício 1. Considere as seguintes funções Matlab

```
function y=f1(x)
perimetro=2*pi*r;
area=pi*r^2;
end

function area=f3(r)
perimetro=2*pi*r;
area=pi*r^2;
end

function [perimetro,area]=f5(r)
perimetro=2*pi*r;
area=pi*r^2;
end
```

```
function y=f2(r)
perimetro=2*pi*r
area=pi*r^2
end

function [area,perimetro]=f4(r)
perimetro=2*pi*r;
area=pi*r^2;
end

function [area,perimetro]=f6(r)
perimetro=2*pi*r;
area=pi*r.^2;
end
```

Qual o resultado das seguintes instruções (use o Matlab apenas para verificar as suas respostas)

a)
```
>> r=1;
>> f1(r)
>> f2(r)
>> f3(r)
```

b)
```
>> area=f3(1)
>> area=f4(1)
>> area=f5(1)
```

c)
```
>> [x,y]=f3(1)
>> [x,y]=f4(1)
>> [area,perimetro]=f5(1)
```

d)
```
>> r=[1,2];
>> [area,perimetro]=f5(r)
>> [area,perimetro]=f6(r)
```

Exercício 2. Execute e comente as seguintes *scripts*.

a)
```
graus=input('Graus? ');
rads=graus/180*pi;
disp([num2str(graus),' graus, corresponde a ',num2str(rads), ' radianos'])
```

b)
```
x=input('x? ');
y(x>0)=1;
y(x<=0)=-1;
disp(y)
```

c)
```
x=1:10;
y=sqrt(x);
fprintf('%3i',x)
fprintf('%3i\n',x)
fprintf('%12.8f\n',x)
fprintf('%12.8f\n',y)
fprintf('%3i  %12.8f\n',[x;y]);
fprintf('  x       sqrt(x) \n'),fprintf(' %3i  %12.8f\n',[x;y]);
```

Exercício 3. Escreva uma pequena *script* ou função para avaliar as seguintes funções:

a)
```
h(T) = T - 10        se 0 < T < 100
     = 0.45 T + 900  se T > 100
```

Casos a testar: (i) T $=$ 5, h = -5; (ii) T $=$ 110, h $=$ 949.5.

b)
```
f(x) = -1     se x < 0
     =  0      se x = 0
     =  1      se x > 0
```

Teste para vários casos e compare os resultados da função **sign** do Matlab.

Exercício 4. Escreva uma função `function int = int_aleat(a,b)` para gerar inteiros aleatórios entre a e b (inclusivé).

Exercício 5. Escreva uma função `function cb = coeff_binom(n,r)` para calcular os coeficientes binomiais $\binom{n}{r}$.

Exercício 6. Escreva uma função `function [elems, mns] = nonzero(A)` que retorna em `elems` todos os elementos não nulos de uma dada matriz `A` e em `mns` a média de todas as colunas de `A`.

Exercício 7. Escreva uma *script* que produza o seguinte resultado:

```
1  Hello world
2  Hello world
3  Hello world
4  Hello world
5  Hello world
6  Hello world
7  Hello world
```

```
 8  Hello world

 1  Hello world    5  Hello world
 2  Hello world    6  Hello world
 3  Hello world    7  Hello world
 4  Hello world    8  Hello world

 1  Hello world    2  Hello world
 3  Hello world    4  Hello world
 5  Hello world    6  Hello world
 7  Hello world    8  Hello world
```


Exercício 8. Em cada uma das seguintes questões, avalie o fragmento de código Matlab apresentado, para cada um dos casos indicados. Use, de seguida, o MATLAB para confirmar as suas respostas.

a)

```
if n > 1
   m = n+1
else
   m = n - 1
end
```

```
(i) n =   7    m = ?
(ii) n =   0    m = ?
(iii) n = -10    m = ?
```

b)

```
if 0 < x < 10
   y = 4*x
else
   y = 500
end
```

```
(i) x = -1   y = ?
(ii) x =  5   y = ?
(iii) x = 20   y = ?
```

c)

```
if z < 5
   w = 2*z
elseif z < 10
   w = 9 - z
elseif z < 100
   w = sqrt(z)
else
   w = z
end
```

```
(i) z =   1    w = ?
(ii) z =   9    w = ?
(iii) z =  60    w = ?
(iv) z = 200    w = ?
```

d)
```
if T < 30              (i) T = 50    h = ?
   h = 2*T + 1        (ii) T = 15    h = ?
elseif T < 10        (iii) T = 0     h = ?
   h = T - 2
else
   h = 0
end
```

e)
```
for i=0:12
   x=pi*i/6;
   disp([x,cos(x)])
end
```

f)
```
n=0;f=1;y=1;
disp('   n          f           y')
while (f < 10000 & y < 10000)
    fprintf('%4i %12.6f %12.6f \n',n,f,y)
    n=n+1;
    f=f*n;
    y=exp(n);
end
```

g)
```
clc
n=input('Nº mecanografico? ');
last=rem(n,10);
disp(' Proximo TPC:')
switch last
    case {4,9}
         disp(' ------------> Exercicios 3 e 4');
    case {5,7}
         disp(' ------------> Exercicios 3 e 5');
    case {3,8}
         disp(' ------------> Exercicios 3 e 6');
    case {2,6}
         disp(' ------------> Exercicios 5 e 7');
    otherwise
         disp(' ------------> Exercicios 6 e 7');
end
pause
dia=magic(4);
mes='Dezembro';
fprintf('\n Data de entrega:  %2i  de %10s de 2007\n',dia(1,end),mes)
```

Exercício 9. Escreva uma função **outraHilb** que, dado o valor de n, construa a matriz de *Hilbert* H, de ordem n, cujos elementos $h_{i,j}$ são da forma $h_{i,j} = 1/(i+j+1)$.

(Nota: Faça `edit hilb.m` e analise a função análoga **hilb** do MATLAB.)

Exercício 10. Dada a matriz A de ordem 4×5, escreva uma *script* para obter a soma de cada coluna de A, usando:

a) a instrução **for**;

b) a função **sum**.

Exercício 11. A seguinte *script* define um vector b, usando um ciclo **for**.

```
% Definir o vector b=(b_i)=sqrt i; i=1,...,10000
% usando um ciclo FOR
%
clear;
tic;
for i=1:10000
   b(i)=sqrt(i);
end
t=toc;
disp(['Tempo de execucao do ciclo FOR e',num2str(t)]);
```

a) Apresente uma versão "vectorizada" desta *script*.

b) Compare o tempo de execução das duas *scripts*.

(Use as funções **tic** e **toc** e execute várias vezes as suas *scripts* para ter uma ideia do tempo médio de execução de cada uma delas.)

Exercício 12. Na tabela seguinte apresenta-se a taxa (%) de IRS a aplicar aos rendimentos de 2007 para residentes no Continente, Madeira e Açores.

Rendimento colectável (euros)	Continente	Madeira	Açores
Até 4 544	10,5	8,5	8,4
Entre 4 544 e 6 873	13,0	11,0	10,4
Entre 6 873 e 17 043	23,5	22,0	18,8
Entre 17 043 e 39 197	34,0	32,5	27,2
Entre 39 127 e 56 807	36,5	36,0	29,2
Entre 56 807 e 61 260	40.0	39,0	32,0
Mais de 61 260	42.0	41.0	33,6

Escreva um programa que, conhecido o rendimento e a residência de um cidadão português, indique qual a taxa de IRS a aplicar ao seu rendimento de 2007.

Exercício 13. Escreva uma *script* que dado um valor fornecido pelo utilizador, da temperatura T_F em graus Fahrenheit, calcule a temperatura equivalente T_C em graus Celsius. (Relembre que $T_F = 1.8T_C + 32$). Esta *script* só deve terminar, quando nenhum valor for introduzido pelo utilizador. (A função **isempty** pode ser útil).

Exercício 14. Escreva um programa que leia uma letra e escreva "Vogal" ou "Consoante" conforme o tipo da letra lida. Se o caracter lido não for uma letra válida, então o programa deve escrever uma mensagem de erro. (As funções **isletter** e **lower** podem ser úteis).

Exercício 15. Escreva uma função **isinteger** tal que `isinteger(n)` é 1 se n é um número inteiro e 0, nos outros casos.

Exercício 16. Escreva uma função para gerar um array aleatório de inteiros entre a e b.

Exercício 17. Chama-se número harmónico a todo o número h_n que possa ser escrito como

$$h_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}, \quad n \in \mathbb{N}.$$

Escreva uma *script* que lhe permita, dado um determinado natural n, encontrar o valor de h_n.

Nota: O inteiro n deve ser pedido interactivamente ao utilizador, através do uso do comando **input**.

Exercício 18.

a) Escreva uma função

`s=somaprogr(r,n)`

para calcular a soma de uma progressão geométrica $1 + r + r^2 + \ldots + r^n$, para r e n variáveis. Teste essa função com os valores de $r = 0.5$ e $n = 10, 20, 100$ e 1000.

b) Faça `help nargin` para obter informação sobre a função pré-definida **nargin**. Utilize **nargin** para poder invocar a sua função apenas com um argumento de entrada, tomando, por defeito, $n = 20$.

Gráficos 2D

Exercício 19. Considere a função $f(x) = \text{sen}(2\pi x)$.

a) Use a função **linspace** para obter uma tabela de valores da função f, em 100 pontos igualmente espaçados do intervalo $[0, 1]$. Use o comando **plot** para esboçar o gráfico da função.

b) Repita a alínea anterior, definindo uma função anónima e usando o comando **fplot**.

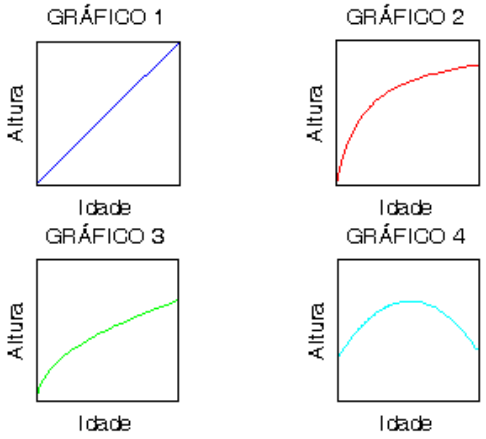
Exercício 20. Desenhe uma circunferência de centro no ponto (2,3) e raio 2, usando a função **plot**, a função **fplot** e a função **ezplot**.

Exercício 21. A ß seguinte desenha dois gráficos da função sen(x^3), no intervalo $[2, 4]$, usando a função **plot** e **fplot**. Execute a ß e interprete os resultados.

```
t=linspace(2,4,50);
y1=sin(t.^3);
subplot(2,1,1)
plot(t,y1,'b')
gtext('Uso da funcao plot');
subplot(2,1,2)
f=@(x) sin(x.^3);
fplot(f,[2,4],'g')
title('Uso da funcao fplot');
```

Exercício 22. Represente, num mesmo gráfico, as funções $f(x) = \sin(4x)$, $g(x) = x\cos(x)$ e $h(x) = (x+1)^{-1}\sqrt{x}$, no intervalo $[1, 10]$, assinalando ainda o ponto $P = (4, 5)$. Use cores e estilos diferentes para cada gráfico e escreva o texto 'ponto isolado' junto do ponto P. Altere depois os eixos, de forma a poder ter uma visão mais pormenorizada da função h.

Exercício 23. Obtenha um gráfico análogo ao seguinte.



Mais exercícios...

Exercício 24. Escreva uma ß que determine se um dado inteiro é divisível por 2 e/ou 3.

Exercício 25. Escreva uma função `teste = lados(a,b,c)` cujos parâmetros de entrada são 3 números reais positivos `a, b` e `c`. Se existir um triângulo cujas medidas dos lados sejam esses números, a função deverá retornar o valor 1 (caso contrário, `teste=0`). Modifique esta função para permitir classificar o triângulo em *equilátero*, *isósceles* ou *escaleno* (no caso `teste=1`).

Exercício 26. Escreva uma função que, dados três pontos $p1$, $p2$ e $p3$, desenhe o triângulo com vértices nesses pontos. A função deverá enviar uma mensagem de erro, caso os pontos não definam um triângulo.

Exercício 27. Complete a seguinte função

```
function meuplot(funcao,dfuncao,lim,ponto)
% MEUPLOT desenha o grafico da funcao FUNCAO, no intervalo  LIM,
%        e da recta tangente ao grafico no ponto de abcissa PONTO
%
%        A equacao da recta tangente e dada por
%        y-funcao(ponto)=dfuncao(ponto)*(x-ponto)
%
%        FUNCAO e DFUNCAO podem ser especificadas como uma
%        funcao anonima ou atraves de uma function handle.
%
% Exemplo:
%         meuplot(@runge,@derivadarunge,[-1 1],0.5)
%
%         f=@(x) 1./(1+25*x.^2), df=@(x) -50*x./((1+25*x.^2)^2)
%         meuplot(f,df,[-1 1],0.5)
%
```

Exercício 28. Determine o maior valor de n para o qual $\sqrt{1^3}+\sqrt{2^3}+\cdots+\sqrt{n^3}$ é menor que 1000.

Exercício 29. Escreva uma função `m=reverso(n)` que, dado um número natural, calcule o número reverso, isto é, o número com os algarismos na ordem contrária à original. (Reveja o Exercício 9 do Capítulo 2.)

Exercício 30. Use a função anterior para escrever uma função **iscapicua** tal que `iscapicua(n)` é 1 se n é uma capicua e 0, nos outros casos.

Exercício 31. Há apenas quatro números naturais diferentes de 1 que podem ser escritos como a soma dos cubos dos seus algarismos. Sabendo que estes números se encontram entre 100 e 999, escreva uma ß para os obter.

Resolva os exercícios seguintes usando uma função ou script. A natureza do input/output, a forma de apresentação dos resultados e o controle das mensagens de erro, quando não explicitados, são deixados ao cuidado do aluno.

Exercício 32. Use a função `coeff_binom(n,r)` do Exercício 5 para obter o seguinte *triângulo*:

$$\binom{1}{1}$$

$$\binom{2}{1} \quad \binom{2}{2}$$

$$\binom{3}{1} \quad \binom{3}{2} \quad \binom{3}{3}$$

$$\binom{4}{1} \quad \binom{4}{2} \quad \binom{4}{3} \quad \binom{4}{4}$$

Exercício 33. Seja $x = (x_1, x_2, \cdots, x_n)$ um vector de n números reais e seja $w = (w_1, w_2, \cdots, w_n)$ um vector de n números reais não negativos tais que $w_1 + w_2 + \cdots + w_n > 0$. O número

$$mp := \frac{\sum_{k=1}^{n} w_k x_k}{\sum_{k=1}^{n} w_k},$$

diz-se a *média pesada* de x e o vector w diz-se o vector dos *pesos* associados a x.

Escreva um programa para, dados os vectores x e w, calcular a média pesada, como definida acima. O programa deve prever as seguintes situações de erro:

- os vectores x e w não têm a mesma dimensão;
- pelo menos um peso é negativo;
- a soma dos pesos é zero.

Exercício 34. Como sabe, os números de Fibonacci são calculados de acordo com a seguinte relação:

$$F_n = F_{n-1} + F_{n-2}, \quad \text{com } F_0 = F_1 = 1$$

a) Obtenha os 10 primeiros números de Fibonacci.

(Use uma implementação diferente da já feita nas aulas, isto é, não recursiva).

b) Considere os rácios $r_n = \frac{F_n}{F_{n-1}}$. Calcule os primeiros 50 rácios.

c) Sabendo que $\lim r_n = \Phi$, onde $\Phi = \frac{1+\sqrt{5}}{2}$ é o famoso número de ouro, pode afirmar que os resultados obtidos ilustram esta propriedade?

Exercício 35. Os *polinómios de Legendre* $\mathcal{P}_n(x)$ são definidos pela seguinte relação de recorrência

$$(n+1)\mathcal{P}_{n+1}(x) - (2n+1)x\mathcal{P}_n(x) + n\mathcal{P}_{n-1}(x) = 0,$$

com $\mathcal{P}_0(x) = 1$ e $\mathcal{P}_1(x) = x$. Calcule os 4 polinómios de Legendre seguintes e represente graficamente os 6 polinómios, no intervalo $[-1, 1]$.

Exercício 36. Considere o seguinte algoritmo que gera uma sequência de inteiros:

- Comece com um inteiro positivo n.
- O próximo número da sequência será $n/2$, se n for par ou $3n + 1$ se n for ímpar.
- Repita este processo com o novo valor obtido, terminando quando atingir 1.

Por exemplo, a seguinte sequência de números será gerada para o valor inicial $n = 22$: `22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1`.

a) Implemente este algoritmo e teste-o para vários valores de n.

b) Para cada n, chama-se *vida de* n ao número de elementos da sequência obtida. Por exemplo, a vida de $n = 22$ é 16. Modifique o programa anterior, de forma a ser possível obter esta informação.

c) Represente graficamente os valores da vida de n, para valores de n entre 2 e 30.

Nota: Este problema é conhecido como Problema de Collatz ou sequência $3n + 1$. É uma conjectura bem conhecida que este algoritmo termina para todo n positivo. (Veja, por exemplo, http://mathworld.wolfram.com/CollatzProblem.html.)

Exercício 37. As regras do Totoloto (`https://www.jogossantacasa.pt/`) são bem conhecidas: de uma tômbola com bolas numeradas de 1 a 49, são extraídas, uma a uma, 6 bolas mais 1 suplementar.

a) Escreva um programa para simular uma extracção do Totoloto. Use um formato de saída de resultados análogo ao usado pela Santa Casa.

Totoloto	Concurso: 01/2008							Data do sorteio: 05/01/2008							
Chave:	26	28	38	39	41	47	+	33	Ordem Saída:	28	26	38	47	41	39	+	33

b) Escreva um programa que dada uma chave (6 números) indique se essa chave é premiada ou não, de acordo com a extracção simulada. No caso de ser uma chave premiada, deve ter em conta que são 5 as categorias de prémios, em função do número de acertos:

1º Prémio - 6 acertos; 2º Prémio - 5 acertos + suplementar; 3º Prémio - 5 acerto;

4º Prémio - 4 acertos; 5º Prémio - 3 acertos.

Anexo A - Alguns comentários...

1. Algumas regras...

A melhor forma de aprender a programar em MATLAB é programando! A análise de programas "bem escritos", como os que o próprio MATLAB apresenta, pode ajudar na difícil tarefa de bem programar.

Apresentamos resumidamente algumas "regras" que pensamos poderão ajudar os alunos a conseguir um estilo de programação adequado aos propósitos de uma primeira disciplina de programação.

① A estrutura geral de uma função deve ser a seguinte:

```
function Parametros_saida=nome_funcao(Parametros_entrada)
%
% Comentarios que especificam a funcao
%

                Codigo da funcao
```

② Os comentários que especificam a função devem incluir todos os detalhes sobre os parâmetros de entrada e de saída.

③ Ao longo da função devem ser incluídos comentários pertinentes que facilitem o entendimento do código. Muitas vezes estes comentários podem ser dispensados se forem usadas variáveis com nomes sugestivos.

④ Deve ser dado especial cuidado à formatação. Todos os ciclos devem ser indentados.

⑤ Recomenda-se um estilo de programação que permita detectar eventuais erros nos parâmetros de entrada.

⑥ Um programa deve estar escrito de forma clara e ter uma organização lógica, de modo a facilitar a sua leitura e entendimento. Além disso, num programa bem estruturado, a detecção de erros é feita com mais facilidade.

⑦ Todos os programas devem ser testados, usando para o efeito exemplos cuja solução seja conhecida.

⑧ A vectorização dos algoritmos deve ser incentivada. Esta técnica tem várias vantagens para além de aumentar a velocidade de execução.

2. Alguns truques!!!

Apresentam-se, de seguida, alguns exemplos de formas de realizar determinadas tarefas. O aluno deverá identificar o objectivo de cada uma dessas tarefas e assinalar, sempre que possível, a solução que lhe parece ser mais "elegante" e rápida.

1.

```
n = 10000;
x = 5 * ones(n,1); %
y = repmat(5,n,1); %
z = zeros(n,1); z(:)=5; %
```

2.

```
n = 1000;
A = 3 * ones(n,n); %
B = repmat(3,n,n); %
```

3.

```
m = 10;
x = 1:5;
A = ones(m,1) * x; %
B = x(ones(m,1),:); %
```

4.

```
n = 5;
x = (1:5)';
A = x * ones(1,n); %
B = x(:,ones(n,1)); %
```

5.

```
for i = 1:10
   t(i) = 2*i;
   y(i) = sin (t(i));%
end

t = 2:2:20;
y = sin (t);%
```

6.

```
n = 10;
A = rand(n,n);
B = ones(n,n);

for k=1:n
   B(2,k) = A(2,k); %
end

B(2,:) = A(2,:);
```

7.

```
for k=1:n
   B(k,1) = A(k,k);
end

B(1:n,1) = diag(A(1:n,1:n));
```

8.

```
x = -250:0.1:250;
for i=1:length(x)
   if (x(i) > 0)
      s(i) = sqrt(x(i));
   else
      s(i) = 0;
   end
end
```

```
x = -250:0.1:250;
s = zeros (size(x));
for i=1:length(x)
   if (x(i) > 0)
      s(i) = sqrt(x(i));
   end
end
```

Anexo B - Algoritmos de ordenação

Objectivo: Ordenar um vector `x` com `n` dados.

Bubble Sort

Descrição

A ideia do *Bubble Sort* é percorrer os dados sequencialmente e, em cada passagem pelos dados, comparar cada elemento com o seu sucessor. Se esses elementos não estiverem ordenados devem ser trocados entre si.

Função bubbleSort

```
function x=bubbleSort(x)
%
% BUBBLESORT ordena o vector x
% Este algoritmo baseia-se no metodo da bolha (bubble sort),
% que consiste em ordenar elementos que estao em posicoes consecutivas

n=length(x);
for k=1:n
    for j=1:n-k
        if x(j)>x(j+1)
            x([j j+1])=x([j+1 j]); % troca dos elementos
        end
        disp(['k= ',num2str(k),'      j= ',num2str(j),'      x= [ ',num2str(x),' ]']);
    end
    disp('-------------------------------------------------')
end
```

Exemplo

```
>> x = [10     0     6    -8    -2     9];
>> bubbleSort(x)
```

```
k= 1      j= 1      x= [ 0  10   6  -8  -2   9 ]
k= 1      j= 2      x= [ 0   6  10  -8  -2   9 ]
k= 1      j= 3      x= [ 0   6  -8  10  -2   9 ]
k= 1      j= 4      x= [ 0   6  -8  -2  10   9 ]
k= 1      j= 5      x= [ 0   6  -8  -2   9  10 ]
-------------------------------------------------
k= 2      j= 1      x= [ 0   6  -8  -2   9  10 ]
k= 2      j= 2      x= [ 0  -8   6  -2   9  10 ]
k= 2      j= 3      x= [ 0  -8  -2   6   9  10 ]
k= 2      j= 4      x= [ 0  -8  -2   6   9  10 ]
-------------------------------------------------
k= 3      j= 1      x= [ -8   0  -2   6   9  10 ]
k= 3      j= 2      x= [ -8  -2   0   6   9  10 ]
k= 3      j= 3      x= [ -8  -2   0   6   9  10 ]
-------------------------------------------------
k= 4      j= 1      x= [ -8  -2   0   6   9  10 ]
k= 4      j= 2      x= [ -8  -2   0   6   9  10 ]
-------------------------------------------------
k= 5      j= 1      x= [ -8  -2   0   6   9  10 ]
-------------------------------------------------
```

Insertion Sort

Descrição

Este método consiste em determinar, para cada elemento que não foi ainda ordenado, qual a sua posição na lista já ordenada e inseri-lo na posição correspondente.

Função insertSort

```
function y = insertSort(x)
%
% INSERTSORT ordena o vector  x
% Este algoritmo baseia-se no metodo da insercao directa (insert sort)
% que consiste em determinar, para cada elemento que nao foi ainda ordenado,
% qual a sua posicao na lista ja ordenada e inseri-lo na posicao correspondente.

y = x(1);
for i =2:length(x)
    y = insert(y, x(i));
     disp(['i= ',num2str(i),'      x= [ ',num2str(y),' ]']);
end
```

```
function b = insert(a, valor)
% INSERT insere um elemento valor numa lista a ja ordenada

b = a;
n = length(a);
while n >= 1 && a(n) > valor
    b(n+1) = a(n);
    n =n - 1;
end
b(n+1)=valor;
```

Exemplo

```
>> x = [10     0     6    -8    -2     9];
>> insertSort(x)

i= 2      x= [ 0  10 ]
i= 3      x= [ 0   6  10 ]
i= 4      x= [ -8   0   6  10 ]
i= 5      x= [ -8  -2   0   6  10 ]
i= 6      x= [ -8  -2   0   6   9  10 ]
-----------------------------------------------------
```

Selection Sort

Este método consiste em encontrar repetidamente o menor elemento e colocá-lo na sua devida posição.

Função selectSort

```
function x=selectSort(x)
% SELECTSORT ordena o vector x
% Este algoritmo baseia-se no metodo da seleccao (select sort),
% que consiste em procurar o elemento minimo e coloca-lo no inicio do vector
% e em seguida fazer o mesmo com os restantes elementos.

n=length(x);
for m=1:n-1
    iminimo=m;
    for k=m+1:n
        if x(k)<x(iminimo)
            iminimo=k;% novo minimo encontrado
        end
    end
    if iminimo>m
        x([m iminimo])=x([iminimo m]); % troca dos elementos
    end
    disp(['m= ',num2str(m),'      x= [ ',num2str(x),' ]']);
end
    disp('-----------------------------------------------------')
end
```

Exemplo

```
>> x = [10     0     6    -8    -2     9];
>>selectSort(x)


m= 1      x= [ -8   0   6  10  -2   9 ]
m= 2      x= [ -8  -2   6  10   0   9 ]
m= 3      x= [ -8  -2   0  10   6   9 ]
m= 4      x= [ -8  -2   0   6  10   9 ]
m= 5      x= [ -8  -2   0   6   9  10 ]
-------------------------------------------------
```

Bibliografia

[HH00] D. J. Higham and N. J. Higham. *MATLAB guide*. SIAM, 2000.

[Mat] The Math Works, Inc., Natwick,MA. *Using MATLAB*.

[Van97] C. F. Van Loan. *Introduction to Scientific Computing*. Prentice Hall, Upper Saddle River, NJ, 1997.